SigC667x HPC Accelerator User Guide

Copyright © 2012 - 2016 Signalogic, Inc.
Revision D2, Feb 2016

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
Tel: 214-349-5551
Fax: 214-343-0163

http://www.signalogic.com
tech_support@signalogic.com

TABLE OF CONTENTS

1 SigC667x Hardware Installation 1

1.1 SigC667x Card Overview 1

1.2 Installing SigC667x Cards 2

1.3 SigC667x Card and Firmware Revisions 6

2 Server Installations 9

2.1 HP DL380p Proliant Series 9

2.2 Dell R720 11

2.3 Supermicro 1U 13

2.4 Small Form-Factor Servers 15

2.5 Desktop and Other PC Enclosures 15

2.6 External Power Cables 16

3 Software Installation 19

3.1 Installation Steps 20

3.1.1 Automated Installation Process Notes 25

3.1.2 Manual Install 26

3.1.3 Tested Linux Distributions and Kernel Versions 26

3.2 Building DirectCore Drivers 28

3.2.1 Loading a DirectCore Driver 29

3.2.2 Linux Kernel-Devel Build Prerequisites 30

3.3 Building DirectCore Libraries 32

3.4 TI Code Generation Tools Download 35

3.4.1 TI RTOS-MCSDK Download 37

3.4.2 TI SYSBIOS, IPC, and XDCtools Download 39

4 Test Programs and Application Demos 40

4.1 memTest 40

4.1.1 memTest Output Description 42

4.1.2 Block Size Parameter Notes 47

4.2 boardTest 48

4.2.1 boardTest Output Description 49

4.2.2 C++ Target 50

4.3 fftTest 51

4.3.1 fftTest Output Description 52

4.4 streamTest 54

4.4.1 streamTest Output Description 55

4.5 Setup for Video Streaming and Image Analytics Demos 58

4.5.1 Tablet Configuration....60
4.5.2 Internet Configuration....61
4.5.3 Router Configuration....63
4.5.4 Running MPlayer....66
4.6 Setup for Transcoding Demos....71

5 VM Operation 72

5.1 VM Add and Configure Hardware....72
5.2 VM Hardware Resource Allocation....80
5.3 VM Remove Hardware....81

6 DirectCore Software 85

6.1 DirectCore Libraries....86
6.1.1 Card Handles and Core Lists....86
6.2 DirectCore Drivers....87
6.2.1 Physical Driver....87
6.2.2 Virtual Driver....87
6.2.3 Driver Installation Notes....88

7 RTAF Software 89

7.1 RTAF Configuration Options....89
7.2 RTAF Peripheral Support....90
7.3 RTAF Development and Host Platform Notes....90
7.3.1 Linux Platform Builds....91
7.3.2 Building without CIM....92
7.3.3 Building with CIM....92

8 CIM® Hyperpiler™ Software 93

8.1 Running User Programs with CIM....93
8.2 vid_streaming demo....94
8.2.1 vid_streaming CIM Build Description....95
8.3 Convolution demo....98

 Copyright © Signalogic, Inc. 2012 - 2016

Revision History

Created	Rev A	Jul 2012	
Modified	Rev B	Nov 2012 – Jan 2013	Merged SigC677x documentation
Modified	Rev B2	Feb 2013	Updated with software changes
Modified	Rev B3	Feb 2013	Added server installations and initial CIM software sections updated with hardware revision and software changes
Modified	Rev B4	Mar 2013	Added video encoding test sections
Modified	Rev B5	Jun 2013	Added RTAF software usage section and updated memTest
Modified	Rev B6	Aug 2013	Added apptest and call test procedures
Modified	Rev B7	Nov 2014	Added fftTest and videoTest DirectCore demo program sections, CIM software install and build details, and CIM® Hyperpiler™ demo section
Modified	Rev B8	Jan 2015	Added 64-core card install instructions, target software install and build instructions, additional driver build notes, and additional library build notes
Modified	Rev B9	Feb 2015	Added additional 64-core card documentation and updated streamTest software demo information
Modified	Rev B9a	Apr 2015	Additional card revision information
Modified	Rev C1	Nov 2015	Added automated install information Removed videoTest reference and replaced with streamTest Added CIM® Hyperpiler™ demo information for vid_streaming and convolution demos

 Copyright © Signalogic, Inc. 2012 - 2016

Modified	Rev C1a	Dec 2015	Added more information about Hyperpiler demos and how to implement image analytics in the streamTest demo
Revised	Rev D	Jan 2016	New software install sections added
Revised	Rev D1	Feb 2016	Fix section numbering. More software install notes
Revised	Rev D2	Feb 2016	Added sections to explain video streaming demo setup, including WiFi router and Android tablets

 Copyright © Signalogic, Inc. 2012 - 2016

1 SigC667x Hardware Installation

The sections below cover installation of specific SigC66xx accelerator cards. Note that section 2, Server Installations, covers a few different types of popular servers and may have some additional hardware installation information that could prove useful, depending on your type of server.

1.1 SigC667x Card Overview

The SigC6678-32 and SigC6678-64 are 32-core and 64-core Gen2 and Gen3 PCIe cards, respectively. These cards combine high performance CPU cores with high bandwidth PCIe interface and large amounts of memory per core, both shared between cores and dedicated external core memory. From the top of the card, the CPU devices can be seen as shown in Figures 1-1a and 1-1b below.

> ***Note:*** ***Production cards have a heat sink and embedded fan that cover most of the top surface of the card.***

Figure 1-1a, SigC6678-32 top view, without heat sink

 Copyright © Signalogic, Inc. 2012 - 2016

Figure 1-1b, SigC6678-64 top view, without heat sink

The SigC6678-32 card has one fan and the SigC6678-64 card has two fans. Including the heat sink and fan(s), the overall card thickness remains within the single-slot height limit given in the PCIe specification.

1.2 Installing SigC667x Cards

To install the SigC6678-32 and SigC6678-64 cards, they should be inserted in a x8 or larger Gen2 or Gen3 PCIe slot. An example of a SigC6678-32 card installed vertically in a x16 PCIe slot is shown in Figure 1-2 below.

 Copyright © Signalogic, Inc. 2012 - 2016

Figure 1-2, SigC667x installed in x8 or larger PCIe slot

Cards may also be installed horizontally, depending on the riser configuration with the host server. Additional airflow from the server is typically not required due to the self-contained heat sink and fan. Maximum power consumption is about 54W for the 32-core card, and about 110W to 120W for the 64-core card.

> ***CAUTION!*** ***Power should be fully removed from the motherboard prior to card installation. Be sure that the card is fully seated and aligned (upon close inspection, no edge connector fingers on the card should be visible above the edges of the PCIe slot), the backplate on the card is not obstructed, the card is fully seated and flat relative to the PCIe connector, and that any part of the card that overhangs the PCIe connector is not touching components on the motherboard, for instance heat sinks or tall capacitors.***

The SigC667x cards support two boot modes: emulation boot mode and I2C boot mode. Boot mode is selected by Switch 1 (labeled "SW1" on the card), as shown in Figure 1-3 below. I2C boot mode is the default setting and should be used for normal SigC667x card operation. Emulation boot mode is reserved for low-level card testing and debug, and is sometimes known as "JTAG debug".

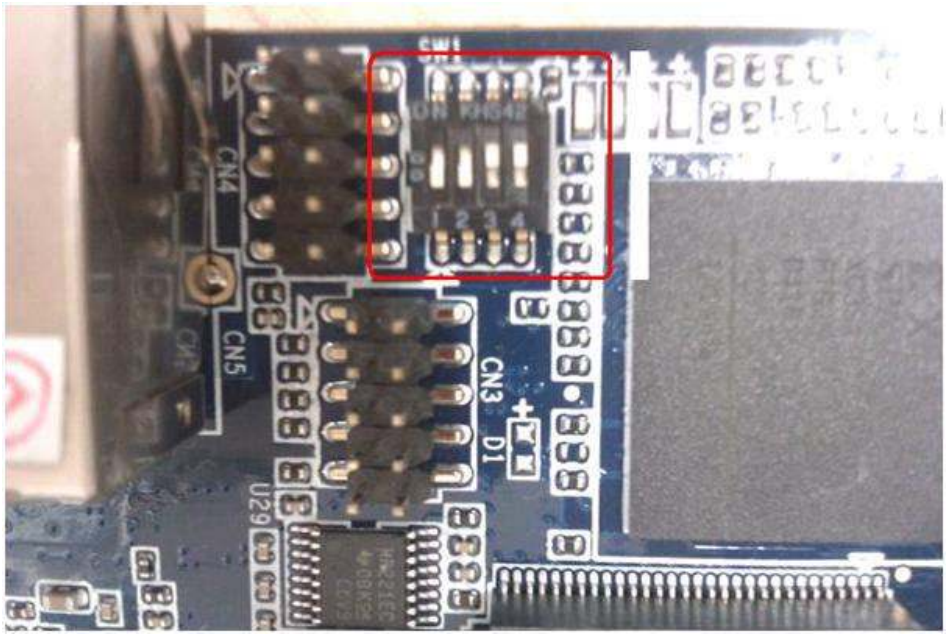

Figure 1-3, SigC6678-32 card SW1 (boot mode)

CAUTION! ***It is a known issue that in some servers SigC667x cards with A101 revision firmware may not complete their on-card I2C boot process before the server completes its internal BIOS scan of the PCIe device tree (which includes PCIe slots). On the SigC6678-32 card there are four (4) "boot LEDs" that will flash to indicate valid PCIe initialization after boot. The placement of these LEDs is shown in Figures 1-4 and 1-5, as shown below.***

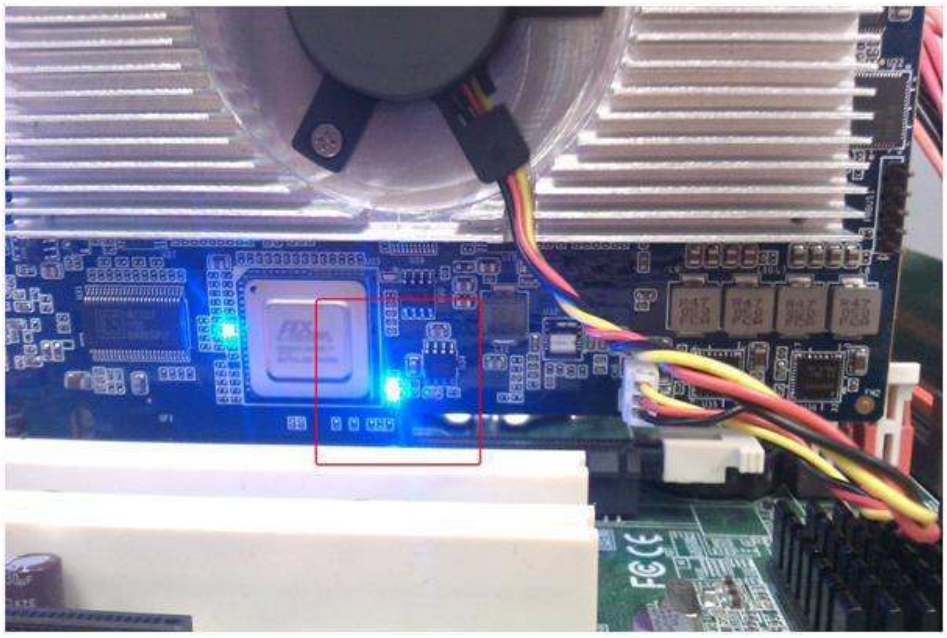

Figure 1-4, SigC6678-32 card PCIe boot LEDs, top side

 Copyright © Signalogic, Inc. 2012 - 2016

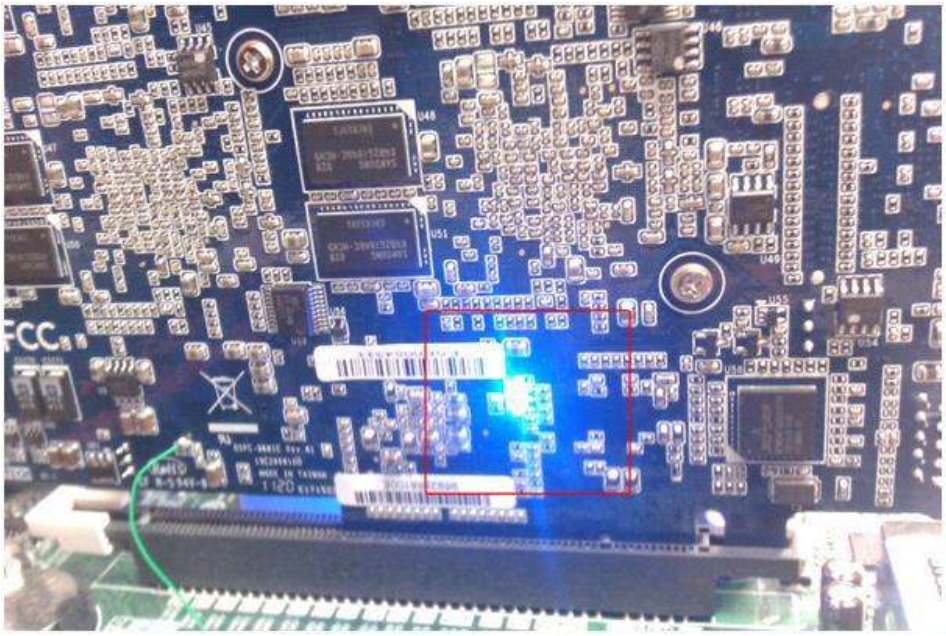

Figure 1-5, SigC6678-32 card PCIe boot LEDs, bottom side

***CAUTION!* If after boot all PCIe LEDs are not flashing, then a system reset, either through software (for example 'shutdown –r now' Linux command) or front-panel switch, can be performed. This is not an issue on cards with A102 revision firmware.**

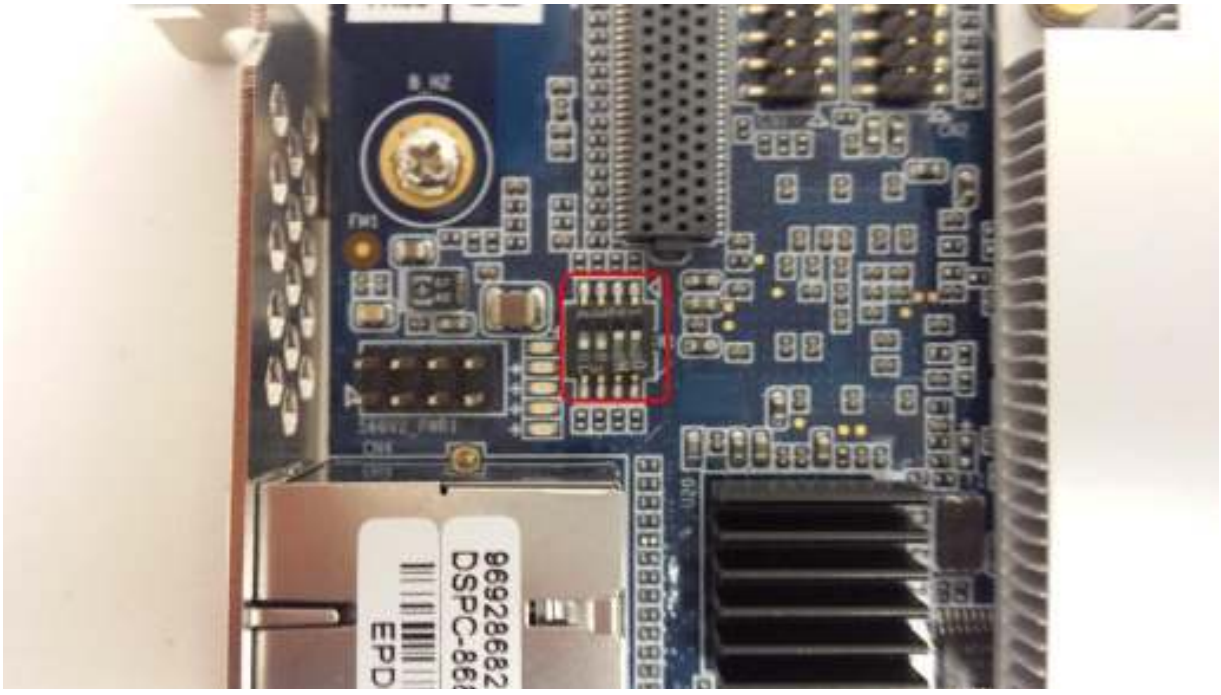

Figure 1-6, SigC6678-64 card DIP switch default settings, top side (highlighted in red)

The SigC6678-64 card DIP switch (labeled "SW1" on the card) should be set as shown in Figure 1-6 above (switch positions 1 and 2 off, and positions 3 and 4 on). If these switch settings are not correct, then the card may fail to boot properly.

Figure 1-7, SigC6678-64 LEDs, top side, after boot (highlighted in red)

After boot, the five (5) LEDs shown in Figure 1-7 above should all be illuminated blue in a "steady on" state, with no blinking (see red highlighted area in the Figure 1-7 image). One LED indicates correct power/voltage conditions on the card, and four (4) LEDs indicate the card's low-level controller FPGA has booted and initialized correctly. As a note, depending on the server form factor, the SigC6678-64, when inserted, may be positioned where the lights and fans are not shown from the top.

CAUTION! If the SigC6678-32 card is inserted horizontally, then make sure the card is flat and is well supported, and does not "sag" or "droop", otherwise the card can develop a permanent warp, which can damage the soldering attachment of chips on the card. For example, if the card is installed on the second slot on a vertical riser, and the PCIe expansion area does not have an internal bracket to support the far end of the card, then it's advisable to insert small pieces of anti-static foam under the card, every 1" to 2", and at the edges of the card, to help support it.

1.3 SigC667x Card and Firmware Revisions

For SigC6678-32, card and firmware revision is printed on the back side of the card, near the inner edge of the PCIe connector, as shown in Figure 1-6 below:

Figure 1-6, SigC6678-32 card and firmware revision information

For SigC6678-64 cards, card and firmware revision information is printed on the back side of the card, as shown in Figure 1-7 below:

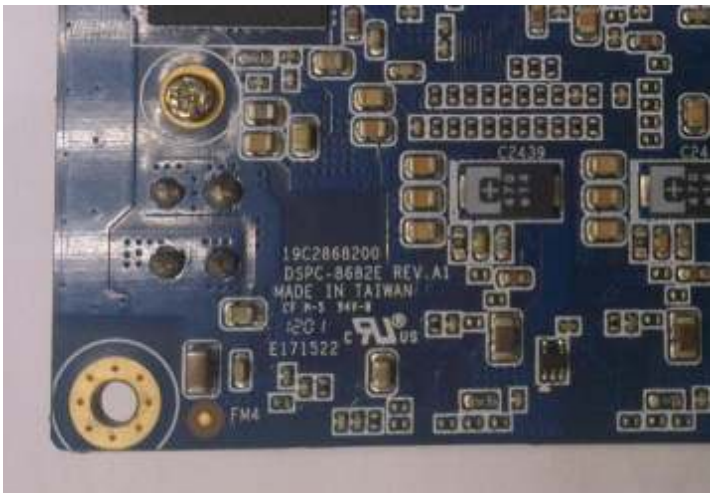

Figure 1-7, SigC6678-64 card and firmware revision information

 Copyright © Signalogic, Inc. 2012 - 2016

Card revision information is given in Table 1-1 below:

Hardware Version Information	**Card Type**	
	Revision **CPU Clock Rate, DDR3 Mem Size**	
Silkscreen labeling on back of card	**SigC6678-32**	**SigC6678-64**
9692868100E or 9692868100	A101 1 GHz [1], 1 GB [2]	
9692868101E	A102 1 GHz, 1 GB	
9692868102E	A103 1 GHz, 1 GB	
9692868122E	A103 1.25 GHz, 2 GB	
19C2868200		A101 1 GHz, 1 GB
19C2868201-01		A102 1 GHz, 1 GB
19C2868202-01		A103 1 GHz, 2 GB
19C2868222-01		A103 1.25 GHz, 2 GB
19C2868203-01		A104-1 1.25 GHz, 2 GB

[1] GByte
[2] GHz, or cycles per sec

Table 1-1, SigC667x card and firmware revision information

In the above table, note that silkscreen labeling includes information about firmware and hardware revision, CPU clock rate, and amount of DDR3 memory per CPU.

 Copyright © Signalogic, Inc. 2012 - 2016

2 Server Installations

The Server Installations section contains information on specific types of servers. This information does not cover all servers, but some of the physical and configuration procedures and requirements documented here may also apply to other server enclosures.

2.1 HP DL380p Proliant Series

Figures 3-1a and 3-1b below show a SigC6678-64 card installed in an HP DL380p Gen8 server. Note the power cable required to connect the card to the 12V connector on the riser.

12V Power Cable

Figure 3-1a, SigC6678-64 card installed in DL380p server, side angle view

Figure 3-1b, SigC6678-64 card installed in DL380p server, top view

 Copyright © Signalogic, Inc. 2012 - 2016

An additional power cable is required for cards with a PEG (PCI Express Graphics) type of connector, which has become a de facto industry standard due to the proliferation of PCIe GPU boards. SigC6678-32 cards do not have a PEG connector, as their power consumption is limited to about 50W. The HP part number of the power cable shown in Figures 3-1a and 3-1b above is:

P/N 504660-003

This HP cable is also shown in Figure 3-7 below (in the picture, the HP cable is on the left, with a bar code tag).

 Copyright © Signalogic, Inc. 2012 - 2016

2.2 Dell R720

Figure 3-2a below shows two (2) SigC6678-64 cards installed in a Dell R720 2U server. Note the 12V power cables connecting cards and risers (Dell P/N 490-13616, also may be available through online distribution using P/N 9H6FV).

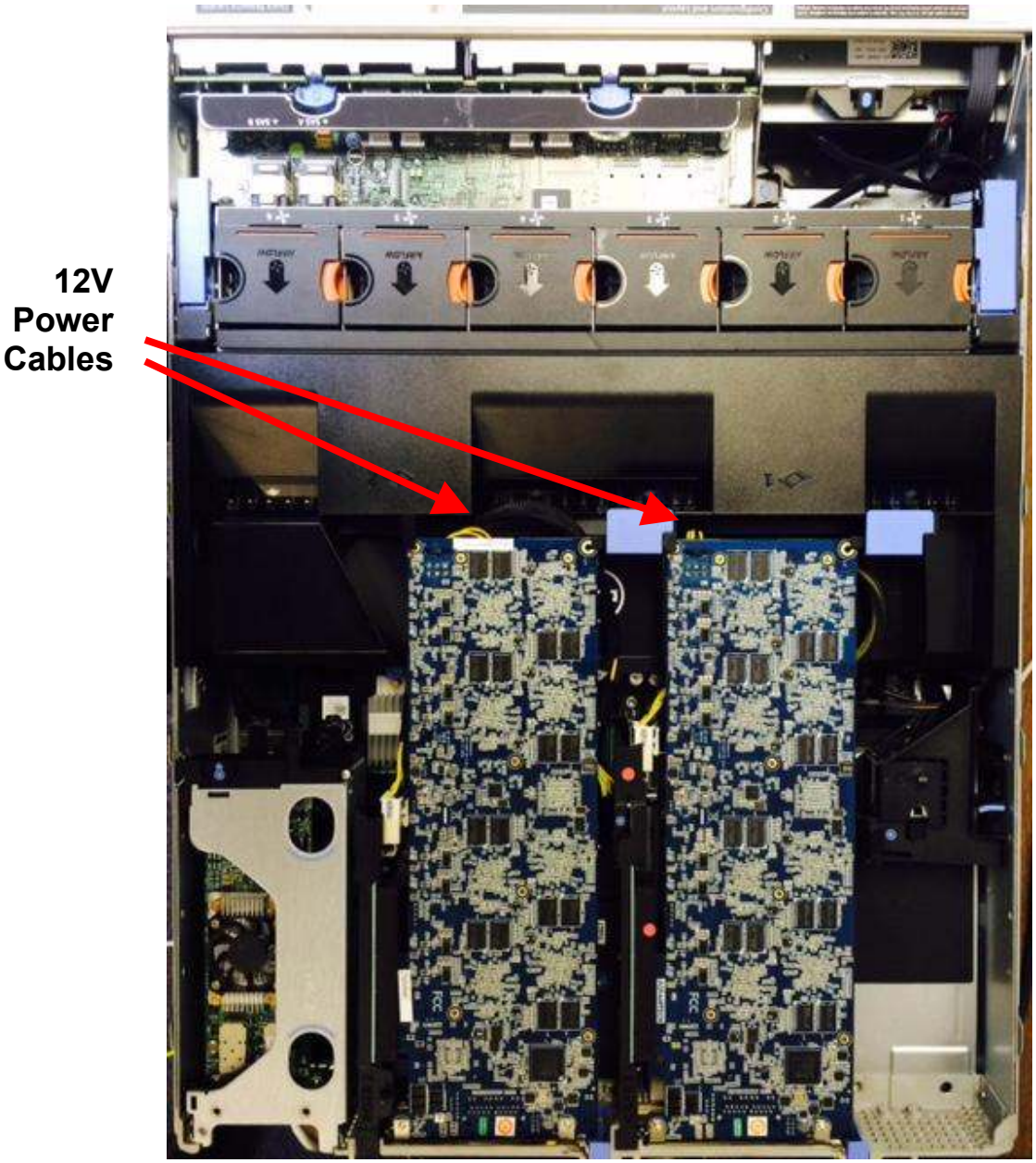

Figure 3-2a, Two (2) SigC6678-64 cards installed in a Dell R720 1U server (top view)

An additional power cable is required for cards with a PEG (PCI Express Graphics) connector, which has become a de facto industry standard due to the proliferation of PCIe GPU boards.

 Copyright © Signalogic, Inc. 2012 - 2016

SigC6678-32 cards do not have a PEG connector, as their power consumption is limited to around 50W. The Dell part number of the power cable shown in Figure 3-1a above is:

P/N 504660-003

Figure 3-2b below shows the SigC667x-64 card power cable adapter used in Dell R720 and R730 servers:

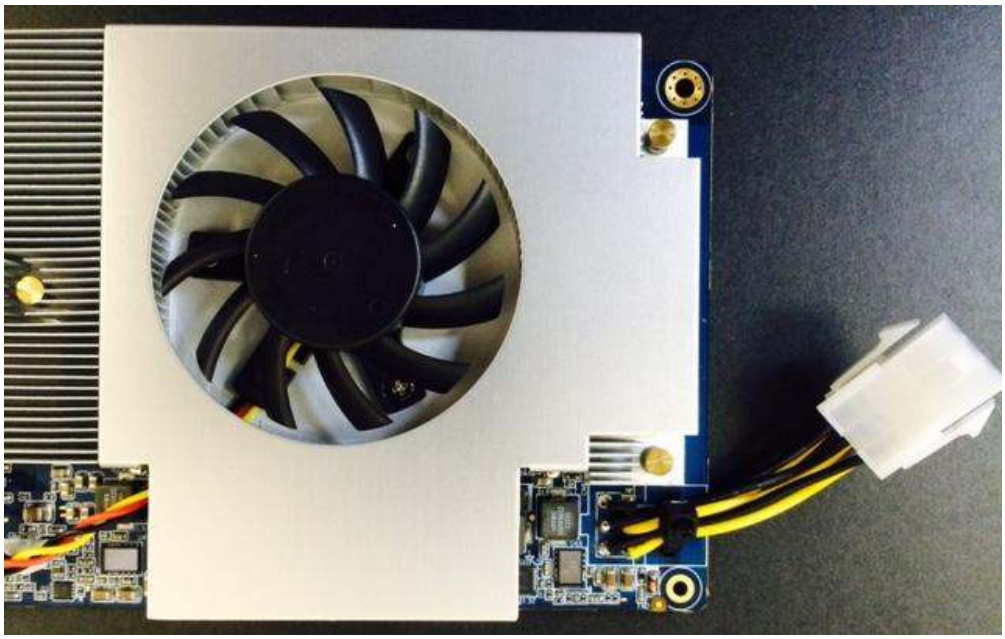

Figure 3-2b, SigC6678-64 card power cable adapter

The power cable adapter mates to the Dell 12V power cable (P/N listed above). Note that in Figure 3-2a above, the power cable adapter is not visible as the cards are installed with component side down.

2.3 Supermicro 1U

Figure 3-3 below shows four (4) SigC6678-32 cards installed in a Supermicro 1U server. In this case, an unusual narrow motherboard and "dual slot riser" configuration makes it possible to insert PCIe cards on both sides of the server. Note that for SigC6678-32 cards, no additional power cables are used.

Figure 3-3, Four (4) SigC6678-32 cards installed in a Supermicro 1U server, top angle view

Figure 3-4 below shows a SigC6678-64 card installed in a 1U server. The card length is about 12.2", which requires a full-length PCIe slot. In this example, the riser card (see annotation in the picture) has two (2) PCIe slots, which allows two (2) SigC6678-64 cards to be installed.

 Copyright © Signalogic, Inc. 2012 - 2016

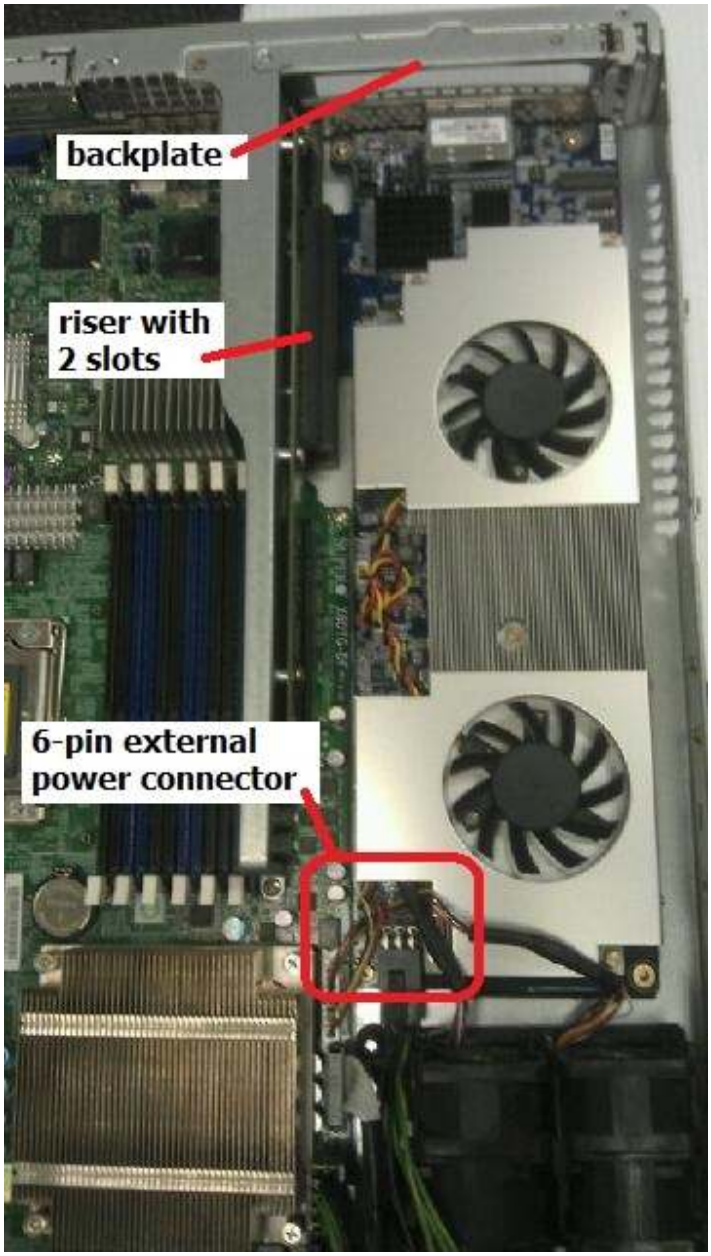

Figure 3-4, SigC6678-64 card installed in a 1U server, top view, with 6-pin external power connection highlighted (red square)

Note that it is also possible to “mix and match” cards, for example, one SigC6678-32 card and one SigC6678-64 card installed on the same riser, or otherwise installed in different PCIe slots in the same server.

2.4 Small Form-Factor Servers

Installation for small form-factor servers, such as mini-ITX, is similar to server installation. Figure 3-5 below shows a dual-core x86 Atom CPU motherboard and a SigC6678-32 card, resulting in an enclosure size of about 8” x 10” x 3” and total power consumption of around 80 W.

Figure 3-5, SigC6678-32 card installed in a mini-ITX server

2.5 Desktop and Other PC Enclosures

Installation for non-server PC enclosures, for example desktop, tower and shuttle cases, is very similar to server installation, with the exception of external power cables for SigC6678-64 cards. Note that for SigC6678-32 cards, no additional power cables are used. See the following sections for more information on external power connection.

2.6 External Power Cables

Figures 3-6 and 3-7 below show different external power cables that can be used with SigC6678-64 cards, for example in "tower" or other traditional PC box enclosures. Note that for SigC6678-32 cards, no additional power cables are used.

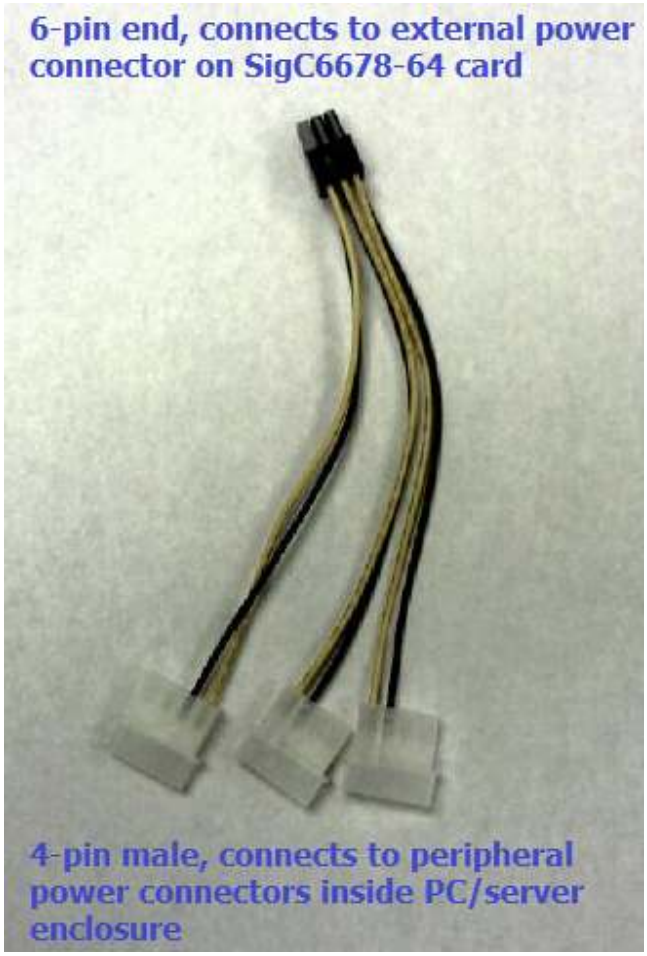
6-pin end, connects to external power connector on SigC6678-64 card
4-pin male, connects to peripheral power connectors inside PC/server enclosure

Figure 3-6, External power cable for SigC6678-64 cards that connects to standard 4-pin peripheral power cables inside a desktop or other non-server form-factor PC enclosure

Figure 3-6 above shows an external power cable that mates with standard 4-pin power cables found inside a desktop or other PC enclosure that provide power to peripherals, such as hard disk drives, DVD players, etc. For the SigC6678-64 card, a minimum of two connectors should be mated, but it is preferable if all three connectors are mated (see notes below).

Figure 3-7 below shows external power cables that mate with a server internal riser card.

Note that typically, yellow wires are +12V and black wires are Gnd. Additionally, some server vendors use proprietary cables connecting to their risers. ***Use caution when sourcing cables, as a generic cable may cause damage to a system.***

 Copyright © Signalogic, Inc. 2012 - 2016

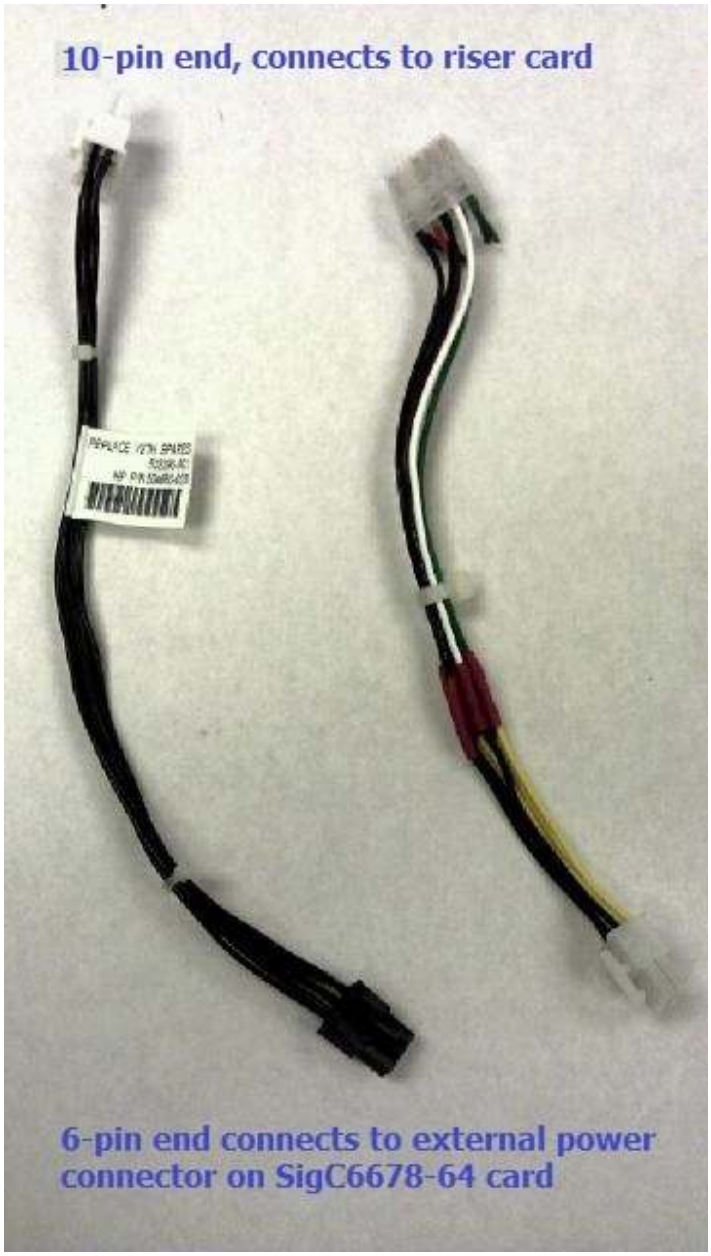

Figure 3-7, External power cable for SigC6678-64 cards that connects to an internal riser card

If for any reason you are unsure about which connector to use, and you want to absolutely verify voltages prior to card installation, a simple test step is to attach the power cable to the server or (or other PC enclosure) internal connector, without the SigC6678-64 card installed and with the top panel or lid open on the enclosure, boot the machine, and use a voltmeter (set to DC volts) to measure the pins. Measured voltages for the external power cable should match up to the pin assignment shown in Figure 3-8 below:

 Copyright © Signalogic, Inc. 2012 - 2016

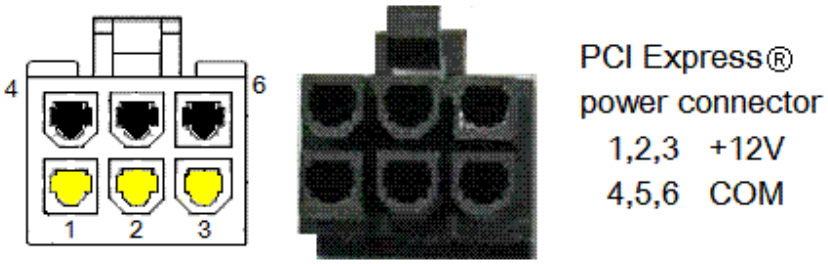

Figure 3-8, 6-pin external power cable pin assignment

Figure 3-8 above shows voltage pin assignment for the 6-pin end of cables shown in above figures and pictures. Notes about Figure 3-8:

- View is facing the connector, i.e. as if you're "standing" on the card, looking at the end of the cable
- COM means Common, or Gnd

Some cables may "break out" the 6 wires, for example Figure 3-6 above shows the 6 wires distributed among 3 peripheral power connectors, in 2-wire pairs (one +12V wire and one Gnd wire for each pair). For the SigC6678-64 card, at least 4 wires (2 yellow, 2 black) should make electrical connection

These connectors are sometimes called PEG connectors (PEG = PCI Express Graphics). Using this terminology may help when ordering cables and/or connectors

3 Software Installation

Signalogic software consists of (i) Linux host software and (ii) target software and build tools. "Host" in this case means the server, ATCA, embedded system, or other platform / enclosure that hosts the "target". Host and target can be physically interfaced in multiple ways:

- Host server shared memory via PCIe bus, for example PCIe cards containing one or more multicore CPUs or other compute devices, such as neural net chips. The SigC6678-32 and SigC6678-64 are accelerator cards containing Texas Instruments c66x series multicore CPUs
- Network connection
- Other methods, such as SRIO

Note the definition of host and target is flexible as there is not always a clear physical boundary. For instance an SoC device might contain one or more ARM cores combined with one more compute intensive cores; in this example the ARM cores would be the host and the compute cores the target. As another example, inside an x86 server, some x86 cores might be "control plane" (host running Linux) and others "data plane" (target).

Below are some general notes on Host Software and Target Software installation and a few important things to keep in mind before proceeding.

Host Software

Host software is installed under the "DirectCore" subfolder, which includes several libraries and drivers, application demo and test programs, and folders such as "include", "common", "lib", etc required to build the application demo and test programs.

Target Software

Target software is installed under a subfolder called mCPU_target, or multicore CPU, which includes different types of possible cores: CPU, DSP, neural net, etc.

Before Proceeding Further

Please note the following requirements before proceeding with software installation:

1. An online distribution location includes several .rar files and an automated install script. The .rar files and auto install script file must stay together in the same folder after downloading
2. You must have the "UnRar" package installed on your system otherwise the script will not be able to extract .rar files and proceed with software installation To locate the correct .rpm file for UnRar, there are links available such as this one https://rpmfind.net/linux/rpm2html/search.php?query=unrar

3.1 Installation Steps

To install Signalogic software, perform the following steps:

1. First, download distribution files from secure page, USB key, or other source as provided. Distribution files include several .rar files and an install script. The download location will include the following subfolders:

   Documentation
   Installation_Scripts
   Media_files (or may be named video_files)
   Software

Figure 3-1 below shows an install location and subfolders in a WinSCP type environment:

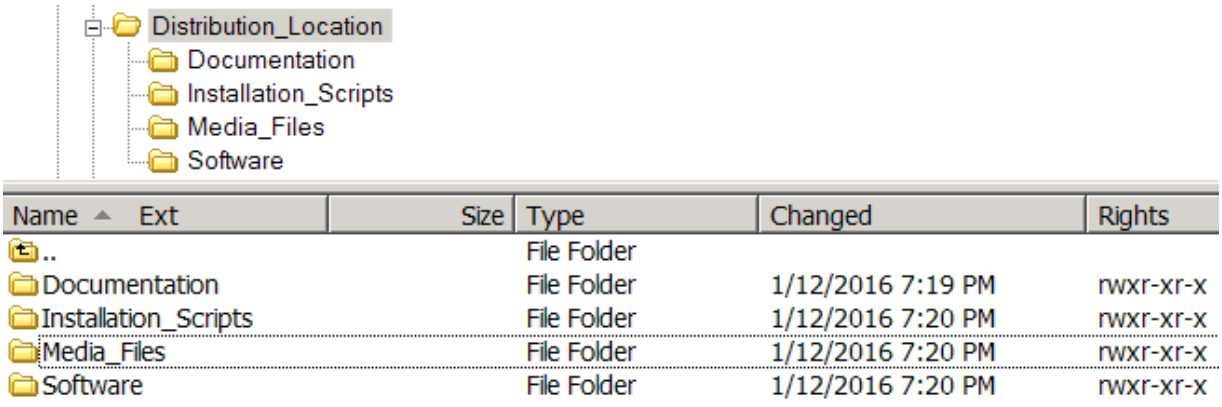

Figure 3-1, Distribution file folder view from a typical download location

Table 3-1 below shows possible .rar file types.

***NOTE!* Depending on which software you have licensed, have on trial, or otherwise have obtained, you may not have all .rar files, or all .rar file contents, as described in Table 3-1 below.**

.rar Files	Type	Description / Comments
DirectCore	Host	Includes drivers (both physical and virtIO), shared libraries, and example application test and demo programs
CIM	Host	Includes CIM® Hyperpiler™ pre-processing tools, utilities and scripts, and demo programs
ffmpeg	Host	ffmpeg support files
Installation_rpms	Host	RPMs possibly required if not found on the target installation

		system
RTAF_target	Target	Real-Time Algorithm Framework (RTAF) components, including run-time test and demo programs and source code. Also includes target components required for CIM (Compute Intensive Multicore) Hyperpiler, OpenCV libs, media libs, and other applications
HPMN_target	Target	HPMN (High Performance Multicore Network) stack components
media_files	Data	Includes .yuv format video files for video compression, streaming, and image analytics demo and test purposes. Also may include audio and speech files (e.g. .wav format files) for streaming and transcoding demo and test purposes
Rose_Boost	Host	ROSE and Boost build tools (used by the CIM Hyperpiler)
TI_Tools	Host	TI build tools

Table 3-1, Distribution .rar file descriptions

Note that .rar files typically have filenames that include a date-time stamp of the form:

DDMMMYY

where DD = day, MM = month, and YY = year. Examples include:

Signalogic_sw_host_DirectCore_22Feb16.rar
Signalogic_sw_target_RTAF_25Feb16.rar

2. Run the automated install script "autoInstall_Sig_SW.sh", which will first prompt as follows:

```
1) Host
2) VM
Please select target for Signalogic SW install [1-2]:
```

After selecting Host or VM target, the install script will prompt as follows:

```
1) Signalogic software installation
2) Uninstall Signalogic software
3) QEMU, libvirt & VMM install
4) TI Tools install
5) ROSE/BOOST install
6) Exit
Please select install operation to perform [1-6]:
```

Here are some notes about the above install options:

- Option 1) will perform a full Signalogic software installation. On choosing this option, the install script will ask for software installation path and then proceed

- Option 2) will perform a software uninstall. This option can be used for a full software uninstall; it may also be useful in case of errors or a need to repeat the installation process
- Option 3) will install modified Qemu, libvirt and VMM to support virtualization of C66x module. Script will check the status of SELinux. If it is in "Enforcing" state, script will prompt to turn it off temporarily as follows:

```
SELinux on your system needs to be disabled
*** WARNING: If SELinux is not turned off, Signalogic c66x
virtualization won't work ***
Do you want to disable SELinux temporarily?
Press [Y]ES or [N]O:
```

NOTE: The script will just turn off SELinux on temporary basis only. System reboot may restart SELinux once again and c66x virtualization will not work. SELinux can be permanently disabled by editing /etc/selinux/config file and set "SELINUX=disabled"

Reference page:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/sect-Security-Enhanced_Linux-Enabling_and_Disabling_SELinux-Disabling_SELinux.html

- Option 4) will install TI Code Generation tools under /opt/ti folder
- Option 5) will install ROSE and BOOST required for CIM source code stream generation process. Installation will take around 3-4 hours of time which is what it will prompt for permission before resuming install as follows:

```
*** WARNING: Approximate time for ROSE/BOOST install is 3-4 Hrs ***
Press [Y]ES or [N]O to continue with the install:
```

- Option 6) will exit the install script

3. After selecting software installation, the install script will ask for an install path. The default is "/". Other tested paths include /home, /opt/, root.

During the install procedure, you may see various prompts but you should not normally see error messages. Upon completion, a Signalogic software folder should be visible, with subfolder hierarchy as shown in Figure 3-2 below.

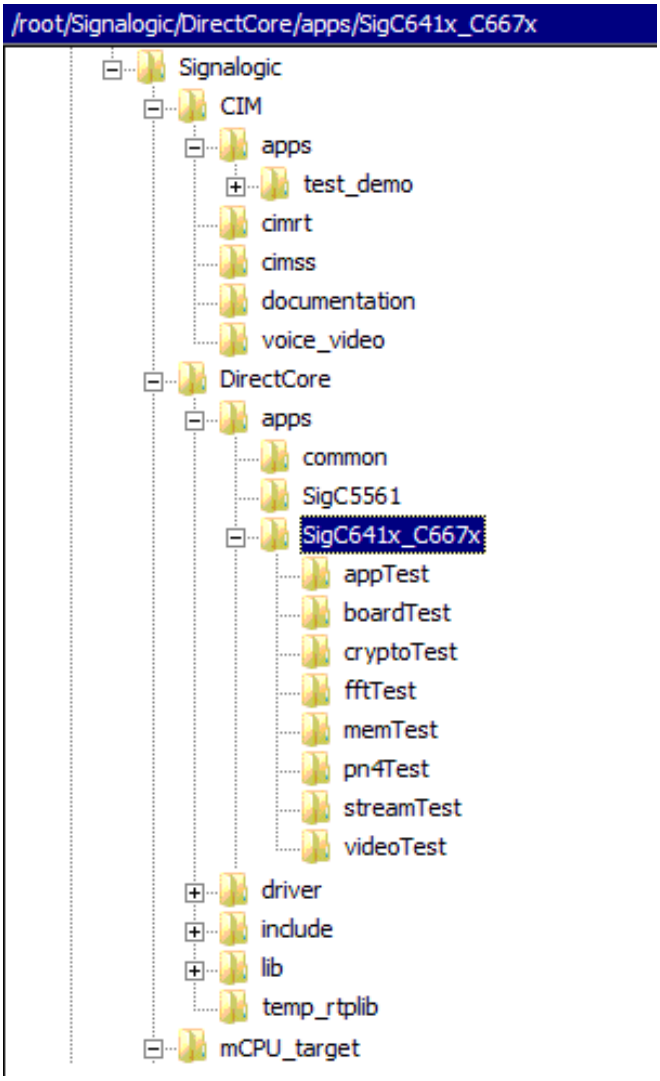

Figure 3-2, Host and target software folder hierarchies after software installation

Here are some notes about the software folder structure:

- In some cases, the Signalogic software folder may be suffixed with "_2014v4", "_2014v5", etc. but if so a symlink will be created that names the folder as "Signalogic"
- In Figure 3-2 above, software has been installed on the "/root" directory. The actual location in your installation will depend on what you entered when prompted for an install path, as mentioned in the automated software steps above

Depending on your version of software, the install procedure may also create a "mCPU_target" folder, which includes source code and libraries for software that runs on accelerator cards. In general, target software includes RTAF components, CIM components for different types of hardware (such as multicore CPU and DSP), and libraries such as OpenCV,neural net, video codecs, and audio and speech codecs. RTAF (Real-Time Algorithm Framework) software provides target software support for host software, including DirectCore software, CIM software,

and user-defined projects. The target .rar file includes source code and other necessary files to build with Texas Instruments command line code generation (build) tools. For more information about RTAF, see section 6 below, RTAF Software. Configuration, project, and other files needed for use with Texas Instruments build tools are also included in the target .rar file.

For SigC667x cards, all TI CPU core code generation and configuration is handled either with makefiles or with the CIM build process, and intended to be used with TI's command line build tools. The software build process does not interact with TI's Code Composer Studio IDE environment, although a CCS mode is supported if needed due to user preference or application-specific reasons.

All target software .rar files will extract into the “mCPU_target” subfolder under the main Signalogic folder, as shown in Figure 3-2 above and Figure 3-3 below.

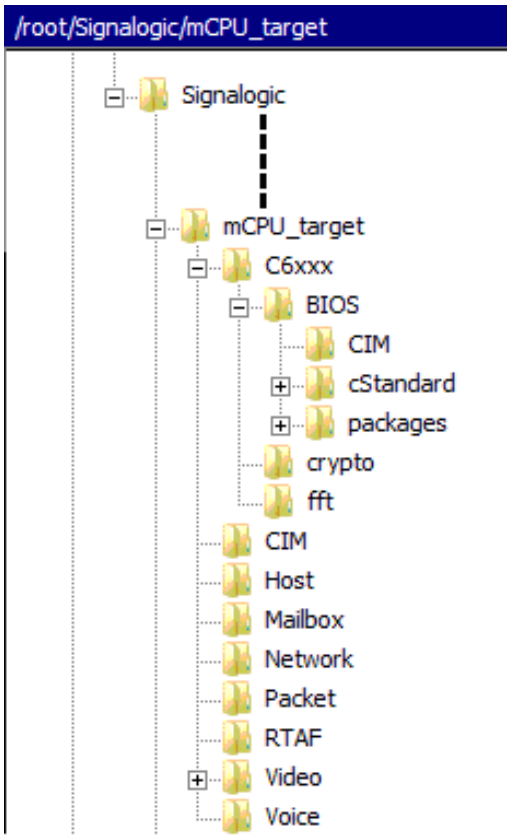

Figure 3-3, Target software folder hierarchy for SigC667x cards

Here are some notes about the mCPU_target software folder:

- The “target” naming convention refers to run-time software for multicore CPU or other PCIe hardware; for example accelerator card, DSP card, video or telecom card, network processing card, neural net card, etc. Supported hardware are referred to throughout the User Guide as "targets", and code that runs on the target as executable target files or "run-time" code

- The mCPU_target folder is required to rebuild executable target files. The mCPU_target folder is not required for application test and demo purposes, as executable target files may be stored on host software folders for run-time purposes
- Source software and libraries required to rebuild target executable files are stored on the mCPU_target folder, including RTAF, CIM, libraries such as OpenCV, video codecs such as H.264 and H.265, and speech/voice codecs

3.1.1 Automated Installation Process Notes

Following are some important notes about the automated installation process.

3.1.1.1 Sudo Root Privilege

At some point the install script will need to temporarily gain sudo root privilege, for example to perform driver configuration, modify config files, etc. When the install script reaches that point it will prompt to perform or skip these steps; if you choose to perform the steps then the script will prompt for a sudo root password.

Note that in Ubuntu, giving a user sudo root privilege can be done by adding the user to the "administrators" group (as shown here http://askubuntu.com/questions/168280/how-do-i-grant-sudo-privileges-to-an-existing-user). In CentOS a user can be added to the "/etc/sudoers" file (as shown here https://wiki.centos.org/TipsAndTricks/BecomingRoot).

3.1.1.2 Dependency Packages

The install script will check for system dependency packages required to complete the installation. If not installed it will install, otherwise it will prompt as follows:

```
Y (Yes)
N (No)
A (yes to All, no further prompts)
```

It should be the case that all required dependency packages will be installed, but it is possible that some may not, depending on your system.

3.1.1.3 Automated Build

Depending on the software license and/or delivery method, the install script will either (i) load pre-built binary files or (ii) build from source code DirectCore libraries, drivers, and example / demo applications. Binary files include .so files for libraries, .ko files for drivers, .out files for multicore CPU targets, and executable files (no extension).

3.1.1.4 Automated Driver Load

The install script will load DirectCore drivers and configure the system so that drivers are loaded automatically after a reboot. After successful software installation, you can verify the main DirectCore driver is loaded using the following command:

```
"lsmod | grep sig_mc_hw"
```

which should show output similar to this:

```
sig_mc_hw       pid          0
```

where “pid” is a Process Id number (for example 92357) and the right hand number is the ”use count” for the driver (which should be zero just after installation).

3.1.2 Manual Install

If for some reason distribution .rar files must be installed manually, they can be unzipped under an arbitrary path using the following commands. This will create a Signalogic software folder, with subfolder hierarchy as shown in Figure 3-2 above. Most files will be installed under the given arbitrary path. Also a “signalogic” configuration file will be installed under:

```
/etc/signalogic
```

Here is a command line example of manual install using the .rar files (actual date-time stamps such as 8Jul15 or 12Feb16 should appear in the filename instead of DDMMYY):

```
unrar x Signalogic_sw_host_DirectCore_DDMMYY.rar /home
unrar x Signalogic_sw_host_CIM_DDMMYY.rar /home
unrar x Signalogic_sw_target_mCPU_target_DDMMYY.rar /home
```

The above commands will unrar files to the “/home” folder. Note the “x” flag is required to preserve .rar file folder structure and perform extraction using full paths.

Note: The folder structure contained in the .rar files must be preserved. After installation, the resulting folder structure should be as shown in Figures 3-2 and 3-3 above.

3.1.3 Tested Linux Distributions and Kernel Versions

Table 3-2 below gives a list of Linux distributions and kernel versions tested with SigC667x cards and DirectCore software.

Linux Distro	Architecture	Kernel Version	Comments
Gentoo	i686	2.6.25	
Wind River 4.3	ppc_32	2.6.34.10	Tested on Advantech ATCA 8901 boards
Red Hat 6.2	x86_64	2.6.32-220	
Red Hat 7.1	x86_64	3.10.0-229	
CentOS 7.1	x86_64	3.15.6	CentOS 7 tested in VMs

 Copyright © Signalogic, Inc. 2012 - 2016

Ubuntu 12.04	x86_64	3.2.0-49-generic	
Ubuntu 14.04	x86_64	3.13.0-32-generic	
Ubuntu 14.10	x86_64	3.16.0	Tested in VMs only
SuSE 12.3	x86_64	3.7.10	

Table 3-2, Tested Linux distributions and kernel versions

3.2 Building DirectCore Drivers

Note: All driver builds and loads are automatically performed during installation using the automated install script. This section is provided to show how to build drivers manually, if for any reason that should be required.

All DirectCore drivers are "Loadable Kernel Modules", or LKMs, which means they are not part of the Linux kernel and can be loaded and unloaded at any time.

Files for host software DirectCore drivers reside on the "driver" and "virt_driver" subfolders under the "DirectCore" subfolder (see Figure 3-2 above). Key files include:

File	Host or VM Instance	Description
`sig_mc_hw.c`	Host	SigC667x driver source code
`sig_virtio.c`	VM	
`Makefile`	Both	Makefile to build and load the driver
`sig_mc_hw.h`	Host	Driver include file (located under ../DirectCore/include)
`sig_virtio.h`	VM	

Table 3-2,DirectCore driver files

The DirectCore unified host driver supports all target hardware with interfaces including PCIe, SRIO, and onchip (SoC). User applications that make DirectCore API calls or incorporate CIM source code generation should contain no user space calls (e.g. ioctl, pread, pwrite, etc) directly to the DirectCore driver; instead, these calls are made by DirectCore libraries via user application API calls. Applications that make user space calls directly to the driver are not supported.

Note 1: To build the DirectCore driver, certain Linux kernel header files are required. These are available if the "kernel-devel" RPM has been installed, and a symlink has been created for "/usr/src/linux". See section 3.2.2 below, Linux Kernel-Devel Build Prerequisites, for more information.

Note 2: In the instructions below, the "#" symbol is used to denote a Linux command-line prompt. It should not be part of any actual command entry.

First go to the driver folder:

```
#cd /install_path/Signalogic/DirectCore/driver
```

To clean the driver folder (optional), enter:

```
#make clean
```

To build the driver, enter:

```
#make
```

3.2.1 Loading a DirectCore Driver

As noted in section 3.1.1.4 above, Automated Driver Load, host and VM instance drivers are automatically loaded by the software installation process, and thereafter upon system reboot or VM restart.

To perform a manual driver load, enter:

```
#cd /install_path/Signalogic/DirectCore/driver
#make
#make load
```

The middle step is optional if the driver has already been built ("make" has already been done at some point, either via automated install or manually).

After loading the driver, you should see screen output similar to Figure 3-4a below:

```
driver$ make load
sync
insmod sig_mc_hw.ko
chmod 666 /dev/sig_mc_hw
../hw_utils/driver_query

Module: sig_mc_hw
   Number of C6678 devices detected: 16
   Number of C6678 cores detected:   128
   Number of reserved C6678 cores:   0

Chip Status:
   Chip 0: Available
   Chip 1: Available
   Chip 2: Available
   Chip 3: Available
   Chip 4: Available
   Chip 5: Available
   Chip 6: Available
   Chip 7: Available
   Chip 8: Available
   Chip 9: Available
   Chip 10: Available
   Chip 11: Available
   Chip 12: Available
   Chip 13: Available
   Chip 14: Available
   Chip 15: Available
```

3-4a, SigC667x manual driver load screen output

To manually unload the driver, enter:

```
#make unload
```

Figure 3-4b below is a screen capture showing files on the driver subfolder and the manual driver unload process:

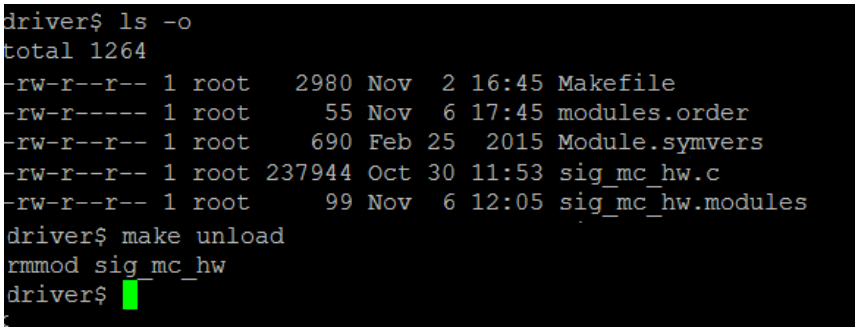
driver$ ls -o
total 1264
-rw-r--r-- 1 root 2980 Nov 2 16:45 Makefile
-rw-r----- 1 root 55 Nov 6 17:45 modules.order
-rw-r--r-- 1 root 690 Feb 25 2015 Module.symvers
-rw-r--r-- 1 root 237944 Oct 30 11:53 sig_mc_hw.c
-rw-r--r-- 1 root 99 Nov 6 12:05 sig_mc_hw.modules
driver$ make unload
rmmod sig_mc_hw
driver$

Figure 3-4b, SigC667x manual driver unload process

The above procedures are identical for host and VM instances. In the case of a host instance, a "physical" or "back end" driver is installed, and in the case of a VM instance a "front end", or "virtual I/O" driver is installed.

3.2.2 Linux Kernel-Devel Build Prerequisites

> ***Note:*** ***All driver builds and loads are automatically performed during installation using the automated install script. This section is provided if for some reason you are building the driver manually.***

When building the DirectCore driver, the "kernel-devel" package, which contains partial Linux source (in particular, certain header files), must be installed. Also, a symlink for "/usr/src/linux" must be created. Instructions for these steps are given below. Note that full kernel source code is not required, as the DirectCore driver is an external driver, not incorporated within the kernel that can be loaded -- and removed -- as needed after the kernel is already booted and running (i.e. loadable kernel module, or LKM).

> ***Note:*** ***The commands below assume that you are logged in as root.***

1. **Step 1.** If you are running a standard Linux kernel you can install the kernel-devel package by following the commands below.

For CentOS and Fedora:

```
[root@host]# yum install kernel-devel.$(uname -r)
```

For Ubuntu and Debian:

```
[root@host]# apt-get install linux-headers-$(uname -r)
```

Note the use of an embedded "uname –r" command in the above command lines. The purpose of this is to ensure installation of the kernel-devel package that **matches** the currently running kernel.

Note that the "uname –r" command can be used separately to determine the version of the currently running kernel:

```
[root@host]# uname -r
```

The result will look similar to this (depending on what kernel revision is installed):

```
2.6.18-400.1.1.el5xen
```

Here are some examples without using an embedded "uname –r" command. Note that these may or may not have exactly correct entry *(comments in italics)*:

```
[root@host]# yum -v install kernel-devel.i686      (for 32-bit CentOS)

[root@host]# yum -v install kernel-devel.x86_64    (for 64-bit CentOS)

[root@host]# apt-get install linux-headers-x86_64  (for 64-bit Ubuntu)
```

2. **Step 2.** In addition to installing the kernel-devel package, a symlink for "/usr/src/linux" must be created, using a command similar to the following:

```
[root@host]# ln -s /usr/src/linux-xx.xx.xx /usr/src/linux
```

where "xx.xx.xx" matches the kernel version returned by the "uname –r" command shown above.

After the symlink is created, the following ls (list) command can be used to verify the symlink (shown as the last line of the ls command result):

```
[root@localhost bin]# ls -l /usr/src/

    total 8

    drwxr-xr-x. 2 root root 4096 Sep 23  2011 debug

    drwxr-xr-x. 5 root root 4096 Apr  8  2014 kernels

    lrwxrwxrwx. 1 root root   34 Nov 13  2012 linux -> kernels/2.6.32-279.14.1.el6.x86_64
```

After these two steps have been performed, the DirectCore driver makefile can be executed, as described in section 3.2 above, "Building DirectCore Drivers".

3.3 Building DirectCore Libraries

***Note:* All library builds are automatically performed during installation using the automated install script. This section is provided to show how to build libraries manually, if for any reason that should be required.**

Source code for DirectCore libraries is located on subfolders under:

```
/install_path/Signalogic/DirectCore/lib/
```

as shown in Figure 3-2, above. If libraries are being built for the first time on a new system, or being rebuilt for a software update on an existing system, the build sequence should be as follows:

Library	Description location in /usr/lib	Type	Comments
enmgr	Engine Manager Library libenmgr	Static	Place holder only for Linux systems
hwmgr	Hardware Manager Library libhwmgr	Static	Not used for Linux systems
hwlib	Hardware Library libhwlib	Shared object	Main hardware interface, containing DSxx APIs
cimlib	CIM Library libcimlib	Shared Object	High level APIs that in turn makes DSxx API calls
filelib	File Library libfilelib	Static	Is used implicitly by hwlib, for example in DSLoadDataFile() and DSSaveDataFile() APIs. Can also be used independently / separately

Table 3-3, DirectCore libraries

For each library, after building, either an archive (.a) or shared object (.so) file should be copied to the default system library directory, typically /usr/lib. Note that the Makefiles do this automatically, but it's always a good idea to verify that library files are being copied; otherwise, application layer programs that use DirectCore APIs may not build correctly.

For each library, use the following build sequence:

```
#cd ../lib/<libname>
#make clean
#make
```

Figure 3-5 below shows some of the DirectCore libraries being rebuilt.

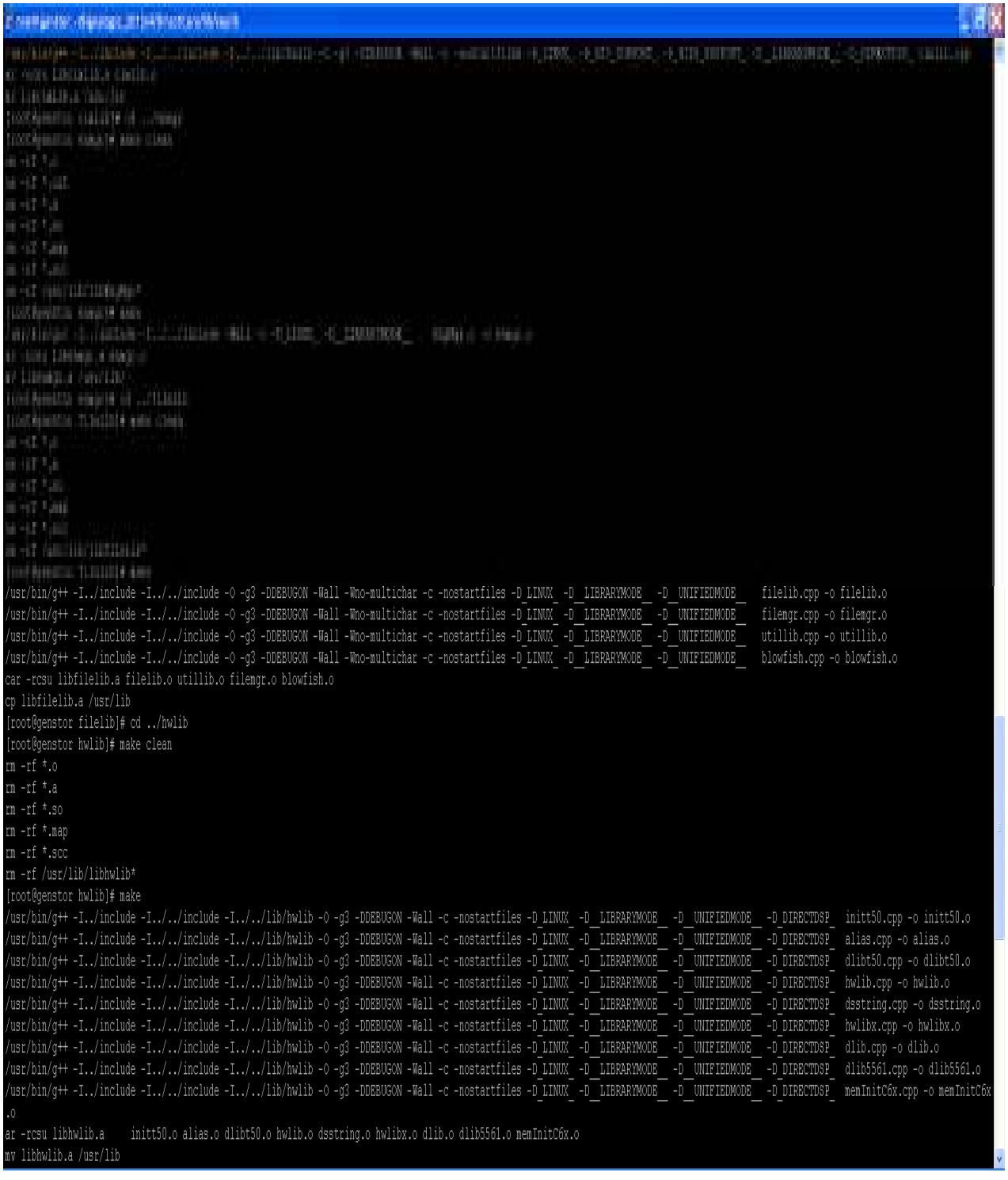
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ filelib.cpp -o filelib.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ filemgr.cpp -o filemgr.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ utillib.cpp -o utillib.o
/usr/bin/g++ -I../include -I../../include -O -g3 -DDEBUGON -Wall -Wno-multichar -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ blowfish.cpp -o blowfish.o
car -rcsu libfilelib.a filelib.o utillib.o filemgr.o blowfish.o
cp libfilelib.a /usr/lib
[root@genstor filelib]# cd ../hwlib
[root@genstor hwlib]# make clean
rm -rf *.o
rm -rf *.a
rm -rf *.so
rm -rf *.map
rm -rf *.scc
rm -rf /usr/lib/libhwlib*
[root@genstor hwlib]# make
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ initt50.cpp -o initt50.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ alias.cpp -o alias.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlibt50.cpp -o dlibt50.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ hwlib.cpp -o hwlib.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dsstring.cpp -o dsstring.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ hwlibx.cpp -o hwlibx.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlib.cpp -o dlib.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ dlib5561.cpp -o dlib5561.o
/usr/bin/g++ -I../include -I../../include -I../../lib/hwlib -O -g3 -DDEBUGON -Wall -c -nostartfiles -D_LINUX_ -D__LIBRARYMODE__ -D__UNIFIEDMODE__ -D_DIRECTDSP_ memInitC6x.cpp -o memInitC6x
.o
ar -rcsu libhwlib.a initt50.o alias.o dlibt50.o hwlib.o dsstring.o hwlibx.o dlib.o dlib5561.o memInitC6x.o
mv libhwlib.a /usr/lib

Figure 3-5, DirectCore library rebuild process

After building each library, resulting library filenames should be (in `/usr/lib/`):

 Copyright © Signalogic, Inc. 2012 - 2016

Default Library Filename	Comments
libenmgr.a	
libhwmgr.a	
libhwlib.so	A static library (".a") version is also created by the default Makefile, and can be used by editing the Makefile to move the resulting .a file to the correct system location, replacing the .so file
libcimlib.so	A static library (".a) version is also created by the default Makefile, and can be used by editing the Makefile to move the resulting .a file to the correct system location, replacing the .so file
libfilelib.a	

3.4 TI Code Generation Tools Download

The software automated install script described in section 3.1 above will automatically install TI code generation and build tools. This section describes the manual procedure to install TI tools, for example if a tools update is required.

The first step is to access the TI Code Generation tools (commonly referred to as CG Tools) download page:

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm

You should see a CG Tools download page similar to Figure 3-6 below:

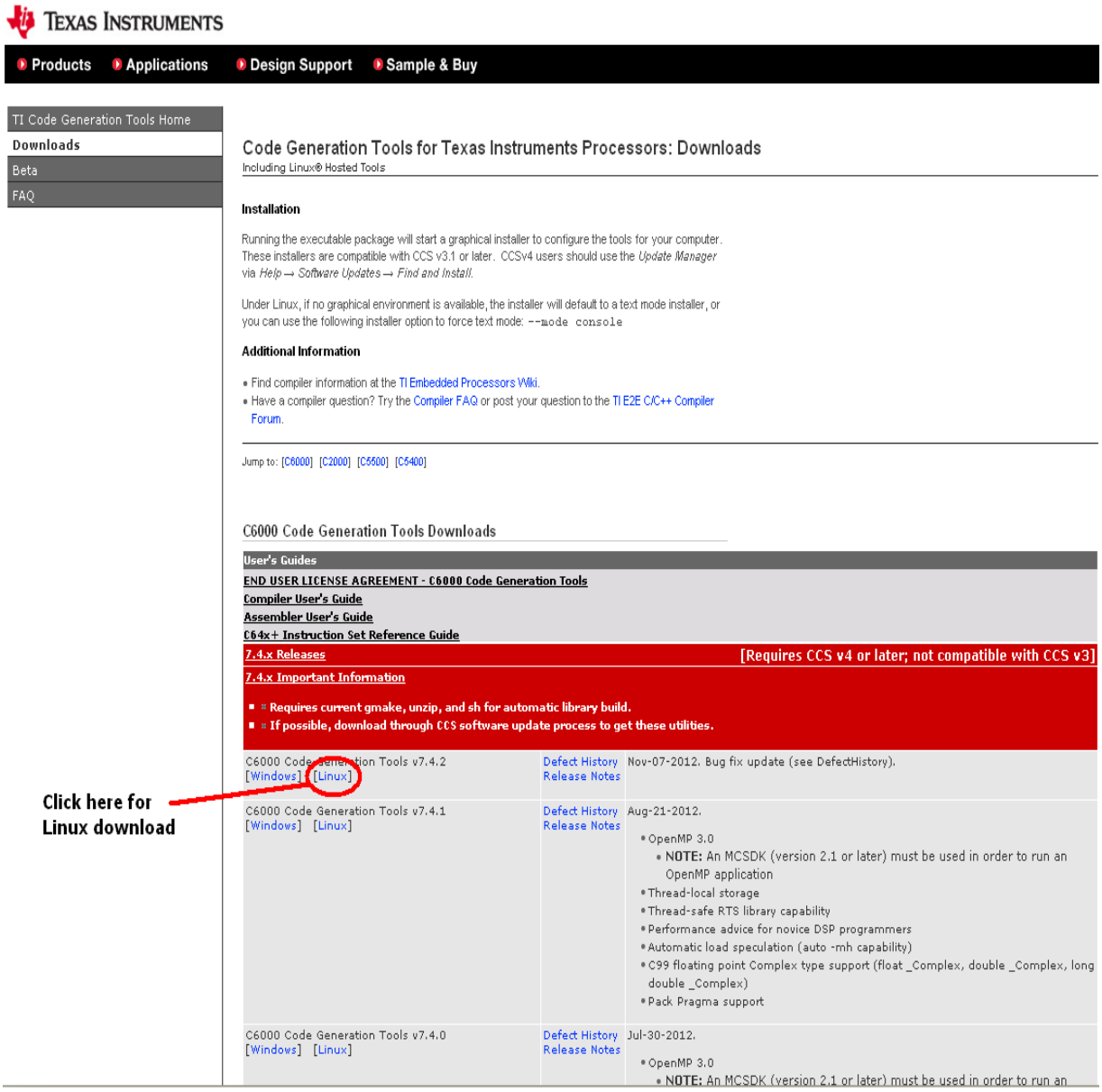
Texas Instruments
Products
Applications
Design Support
Sample & Buy
TI Code Generation Tools Home
Downloads
Beta
FAQ
Code Generation Tools for Texas Instruments Processors: Downloads
Including Linux® Hosted Tools
Installation
Running the executable package will start a graphical installer to configure the tools for your computer. These installers are compatible with CCS v3.1 or later. CCSv4 users should use the Update Manager via Help → Software Updates → Find and Install.
Under Linux, if no graphical environment is available, the installer will default to a text mode installer, or you can use the following installer option to force text mode: --mode console
Additional Information
• Find compiler information at the TI Embedded Processors Wiki.
• Have a compiler question? Try the Compiler FAQ or post your question to the TI E2E C/C++ Compiler Forum.
Jump to: [C6000] [C2000] [C5500] [C5400]
C6000 Code Generation Tools Downloads
User's Guides
END USER LICENSE AGREEMENT - C6000 Code Generation Tools
Compiler User's Guide
Assembler User's Guide
C64x+ Instruction Set Reference Guide
7.4.x Releases
[Requires CCS v4 or later; not compatible with CCS v3]
7.4.x Important Information
▪ Requires current gmake, unzip, and sh for automatic library build.
▪ If possible, download through CCS software update process to get these utilities.
C6000 Code Generation Tools v7.4.2 [Windows] [Linux]
Defect History Release Notes
Nov-07-2012. Bug fix update (see DefectHistory).
C6000 Code Generation Tools v7.4.1 [Windows] [Linux]
Defect History Release Notes
Aug-21-2012.
• OpenMP 3.0
• NOTE: An MCSDK (version 2.1 or later) must be used in order to run an OpenMP application
• Thread-local storage
• Thread-safe RTS library capability
• Performance advice for novice DSP programmers
• Automatic load speculation (auto -mh capability)
• C99 floating point Complex type support (float _Complex, double _Complex, long double _Complex)
• Pack Pragma support
C6000 Code Generation Tools v7.4.0 [Windows] [Linux]
Defect History Release Notes
Jul-30-2012.
• OpenMP 3.0
• NOTE: An MCSDK (version 2.1 or later) must be used in order to run an
Click here for Linux download

Figure 3-6, TI Code Generation Tools download page

Note that this page *may in some cases* require a log-in; if so then you have to register with TI for an account (if you don't already have one). For more information about TI accounts, go to this TI page:

http://processors.wiki.ti.com/index.php/Compiler_Releases

Some notes about the above wiki page:

1. If you click under "Free Compiler Downloads" (on link labeled "Here"), it will take you to the CG Tools download page (Figure 3-6 above).
2. You do not need to have Code Composer Studio (CCS) installed.

Once you have arrived at the TI download page shown in Figure 3-6, download the following TI files to /home/<user>/Downloads:

ti_cgt_c6000_8.0.3_linux_installer_x86.bin
bios_mcsdk_02_01_02_06_setuplinux.bin

```
cd /home/<user>/Downloads
chmod 777 ti_cgt_c6000_8.0.3_linux_installer_x86.bin
chmod 777 bios_mcsdk_02_01_02_06_setuplinux.bin
mkdir /opt/ti     (you will be prompted for a path to install, use this path)

./bios_mcsdk_02_01_02_06_setuplinux.bin
./ti_cgt_c6000_8.0.3_linux_installer_x86.bin
```

Install into the /opt/ti/ directory when prompted for install path.

3.4.1 TI RTOS-MCSDK Download

The second step in a manual TI tools install is to download TI's TI-RTOS-MCSDK, which will download the following components:

- SYSBIOS (Real-time operating system for TI multicore CPUs)
- IPC (Interprocessor Communication)
- XDCtools (utilities and libraries)
- PDK (Platform Development Kit)

To download the TI-RTOS-MCSDK, go to the following page:

http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/latest/index_FDS.html

and locate the product download link with the following label:

```
bios_mcsdk_02_01_02_06_setuplinux.bin
```

and click on that to start the download. Note that the version number portion of the label will change over time (in the link above, this is v2.01.02.06).

Notes about SYSBIOS, IPC, and XDCtools:

1. The version of SYSBIOS installed should be 6.32.05.54 or higher. The version of XDCtools installed should be 3.22.04.46 or higher. These version numbers have been tested in Signalogic labs, although older versions may still work

2. IPC (Interprocessor Communication) is currently not used by DirectCore and CIM software, but may be in the future. For C667x devices, IPC includes SRIO and HyperLink. The SigC6678 cards have SRIO connections between all C667x devices. SRIO is a fast way to move data between C667x processors. HyperLink is also fast, but connects only pairs of devices together

3. PDK (Platform Development Kit) is installed with TI-RTOS-MCSDK. PDK is currently used by DirectCore and CIM software. Also, PDK includes TI' CSL (Chip Support Library), which is not used currently DirectCore and CIM software, but is likely to be used in the future

After downloading and installing TI tools as described in preceding sections, you should have a TI software folder hierarchy similar to that shown in Figure 3-7 below.

Note: The default location for TI software is the /opt/ti folder, at the same level as /root, /home, etc. This is TI's recommended default, and currently the only path supported in this document.

 Copyright © Signalogic, Inc. 2012 - 2016

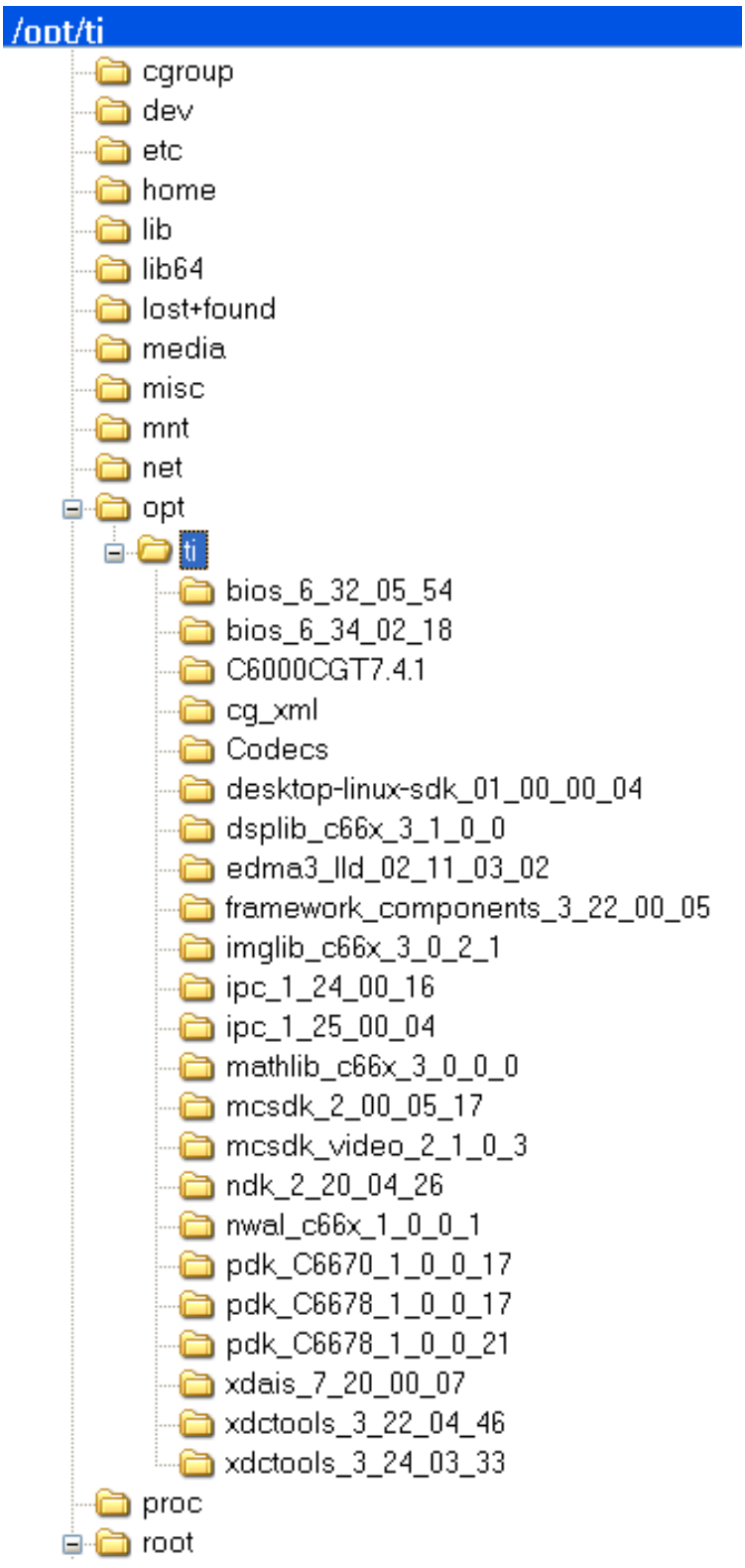

Figure 3-7, TI software folder hierarchy, including Code Gen tools, SYSBIOS, IPC, XDCtools, and PDK

 Copyright © Signalogic, Inc. 2012 - 2016

3.4.2 TI SYSBIOS, IPC, and XDCtools Download

If needed, SYSBIOS, IPC, and XDCtools can be downloaded separately, for example to upgrade one of the components without downloading all of TI-RTOS-MCSDK. To download components separately, go to the following page:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/index.html

Note that currently, PDK cannot be downloaded separately, and must be obtained via TI-RTOS-MCSDK download.

 Copyright © Signalogic, Inc. 2012 - 2016

4 Test Programs and Application Demos

The following test programs are available after default software installation.

Test Program	Card Connections Required	Comments
memTest	None: No loopback connections or other hardware connections are required for this tests	Can be used to validate the card and driver installation. No DirectCore APIs are used, and only executable code for purposes of one-time initialization runs on the target. This is the most minimal test available
boardTest	None: No loopback connections or other hardware connections are required for these tests. The demo "simulates" analog input using interrupt driven executable code running on the target, but no actual physical connection is required	Can be used to validate the card, driver, and library installation
fftTest	None	Example of algorithm parallelization and acceleration
streamTest	Ethernet	Example of high performance video transcoding and streaming. UDP/RTP streaming output can be routed to a mobile device, for example an Android tablet running MPlayer or VideoLAN VLC players
vid_streaming conv	None	CIM® software demos, showing example OpenMP superset pragmas in C source code, source code stream separation and generation, and run-time execution on the target hardware (note -- this example is covered in sections 4.7.2 and 4.7.3, vid_streaming and convolution demos)

4.1 memTest

***Note:** this section assumes you have already built and loaded the DirectCore target multicore CPU driver, as explained in section 3.2 above, Building DirectCore Drivers.*

 Copyright © Signalogic, Inc. 2012 - 2016

The purpose of memTest is to test hardware and driver software directly and at the lowest possible level, with no libraries and minimal user space code. This is not the normal or recommended method of using SigC667x cards but serves instead as a diagnostic and measurement tool. The memTest program has several test modes as follows:

- diagnostic test
- random block test
- broadcast test
- performance test

To build memTest, enter:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C667x/memTest

#make
```

To run memTest, first enter:

```
#./memTest -h
```

This will bring up the help menu showing command line options that can be used to configure the test program and SigC667x hardware.

memTest command line options are shown below ('+' indicates mandatory entry):

-c<*string*>	+	Card designator, e.g. *-cSIGC66XX* or *-cSIGC64XX*
-B<*num*>		Block size
-d		Diagnostic test (default if *-d*, *-r*, or *-z* not entered)
-D		Direct mode (access card registers directly)
-m<*num*>	+	Bitmask for core list (1 = core0, 3 = cores 0 and 1, etc)
-M<*string*>		Memory region for SigC667x, enter *-Moc*, *-Mos*, or *-Me* (onchip mem, onchip shared mem, or external mem)
-n<*num*>		Number of loops (otherwise runs until keyboard exit)
-p<*char*>		Performance measurement, enter *-p* or *-pt* (*-pt* measures data transfer time only)
-P		Master push mode (wait for transfer to complete)

```
-r              Random address block test
-R              Read only
-W              Write only
-z              Broadcast test
```

<num> entries may be specified in either decimal or hex format; if hex, then use a '0x' prefix.

Note that -d, -r, and -z are the possible test types; if none are entered, then the default test type is diagnostic (-d). If more than one test type is entered, then tests are run in the following sequence:

1. diagnostic
2. broadcast
3. random block

Note that diagnostic test runs until a key is pressed, so if multiple test types are entered including diagnostic, then you must exit the diagnostic test (press a key) in order to complete all tests.

Here are sample command lines to run memTest for cores 0 and 1, using the diagnostic test type:

```
#./memTest -m3 -d -cSIGC66XX
```

4.1.1 memTest Output Description

Figure 4-6 below is a screen capture of a successful memTest run for 3 cores by running the following command:

```
#./memTest -m7 -d -cSIGC66XX
```

The screen capture in Figure 4-1 below shows an incrementing loop counter at the top of the terminal display and the error counters for the first 3 cores remain zero as the loop counter value increases. The remaining error counters represent cores not included in the "-m" "core list" parameter given in the command line. These are shown as "n/a" to indicate they are not subject to the ongoing test.

To exit from the program, press any key.

Note! For SigC667x hardware, "core list" with multiple bits set may span one or more target multicore CPUs

 Copyright © Signalogic, Inc. 2012 - 2016

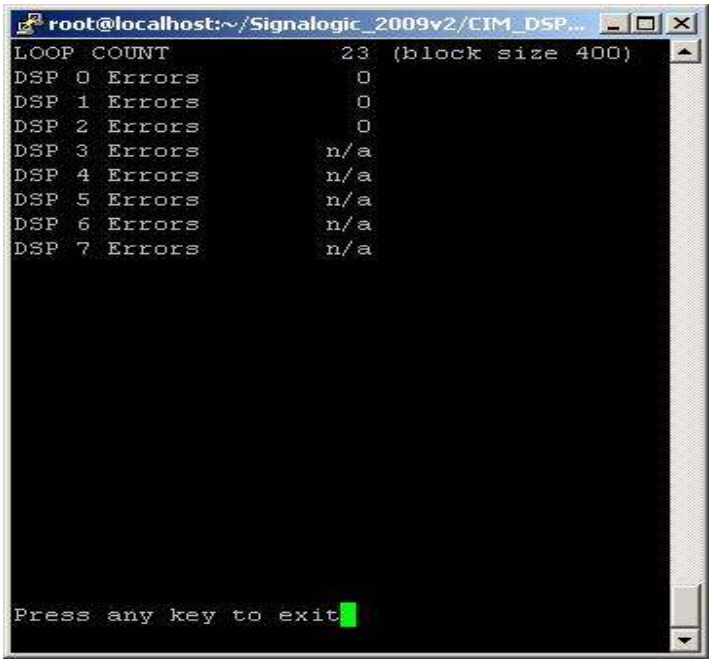
root@localhost:~/Signalogic_2009v2/CIM_DSP...
LOOP COUNT 23 (block size 400)
DSP 0 Errors 0
DSP 1 Errors 0
DSP 2 Errors 0
DSP 3 Errors n/a
DSP 4 Errors n/a
DSP 5 Errors n/a
DSP 6 Errors n/a
DSP 7 Errors n/a
Press any key to exit

Figure 4-1, memTest running diagnostic test for 3 cores on SigC667x hardware

Figure 4-2 below is a screen capture of a memTest diagnostic run for 32 cores, using SigC667x hardware, by executing the following command:

```
#./memTest -cSIGC66XX -d -m0xffffffff -B98304 -P
```

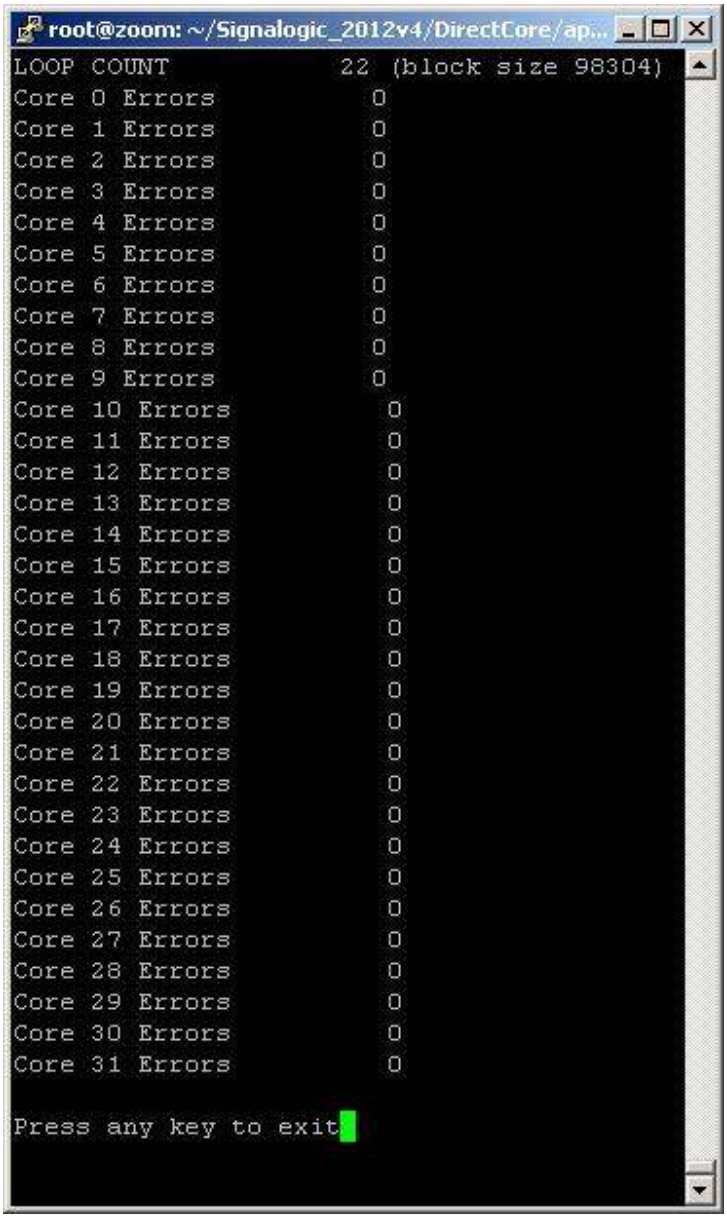
root@zoom: ~/Signalogic_2012v4/DirectCore/ap...
LOOP COUNT 22 (block size 98304)
Core 0 Errors 0
Core 1 Errors 0
Core 2 Errors 0
Core 3 Errors 0
Core 4 Errors 0
Core 5 Errors 0
Core 6 Errors 0
Core 7 Errors 0
Core 8 Errors 0
Core 9 Errors 0
Core 10 Errors 0
Core 11 Errors 0
Core 12 Errors 0
Core 13 Errors 0
Core 14 Errors 0
Core 15 Errors 0
Core 16 Errors 0
Core 17 Errors 0
Core 18 Errors 0
Core 19 Errors 0
Core 20 Errors 0
Core 21 Errors 0
Core 22 Errors 0
Core 23 Errors 0
Core 24 Errors 0
Core 25 Errors 0
Core 26 Errors 0
Core 27 Errors 0
Core 28 Errors 0
Core 29 Errors 0
Core 30 Errors 0
Core 31 Errors 0
Press any key to exit

Figure 4-3, memTest running diagnostic test for 32 cores on a SigC667x card

Figure 4-3 below is a screen capture of a successful random block memTest run for 3 cores spanning two different CPUs, by executing the following command:

```
#./memTest -m0xa8 -r-B204 -n20 -cSIGC66XX
```

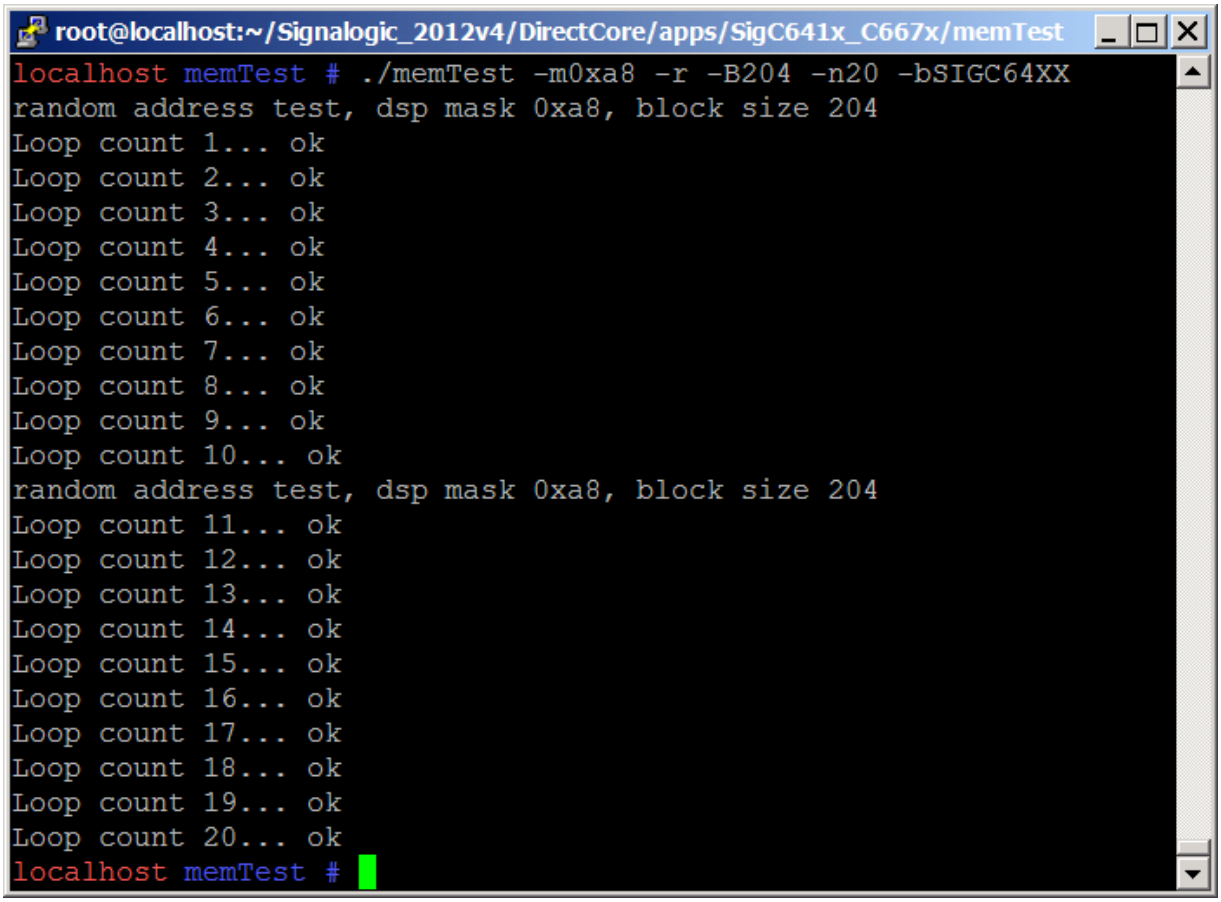
root@localhost:~/Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest
localhost memTest # ./memTest -m0xa8 -r -B204 -n20 -bSIGC64XX
random address test, dsp mask 0xa8, block size 204
Loop count 1... ok
Loop count 2... ok
Loop count 3... ok
Loop count 4... ok
Loop count 5... ok
Loop count 6... ok
Loop count 7... ok
Loop count 8... ok
Loop count 9... ok
Loop count 10... ok
random address test, dsp mask 0xa8, block size 204
Loop count 11... ok
Loop count 12... ok
Loop count 13... ok
Loop count 14... ok
Loop count 15... ok
Loop count 16... ok
Loop count 17... ok
Loop count 18... ok
Loop count 19... ok
Loop count 20... ok
localhost memTest #

Figure 4-4, memTest running random block test for 3 cores on a SigC667x card

Note that in the above command, the `-n20` parameter specifies the test to run for 20 iterations. Without the `-n` parameter the test runs for one iteration by default.

Figure 4-5 below is a screen capture of a successful random block, write-only, test of onchip core memory (i.e. core-specific L2 memory) for 32 cores by executing the following command:

```
#./memTest -cSIGC66XX -B98304 -pt -P -r -m0xffffffff -Moc -n10 -W
```

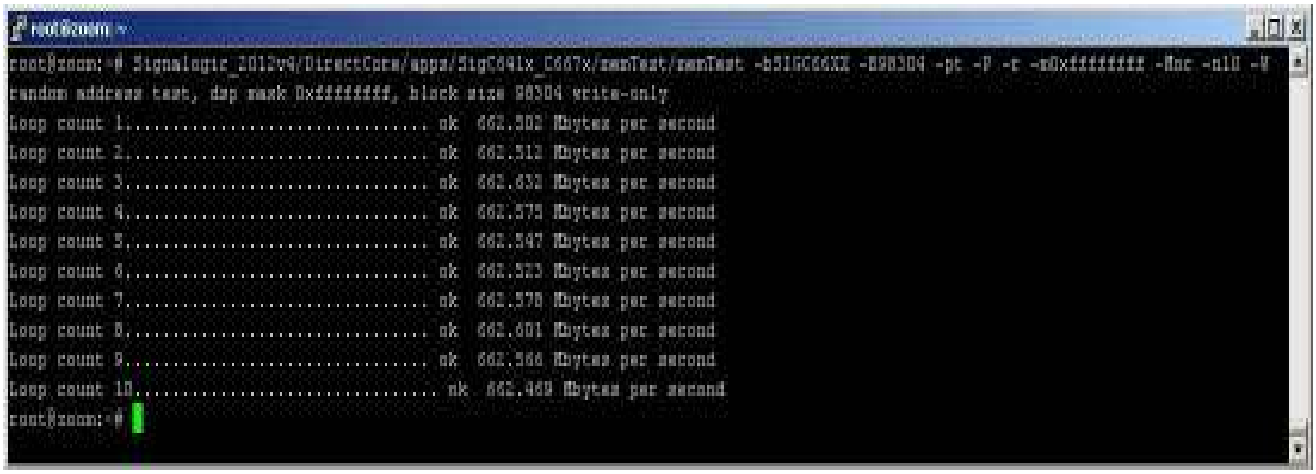
root@zoom: ~
root@zoom: # Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest/memTest -bSIGC66XX -B80304 -pt -P -r -m0xffffffff -Moc -n10 -W
random address test, dap mask 0xffffffff, block size 80304 write-only
Loop count 1..... ok 662.302 Mbytes per second
Loop count 2..... ok 662.513 Mbytes per second
Loop count 3..... ok 662.632 Mbytes per second
Loop count 4..... ok 662.575 Mbytes per second
Loop count 5..... ok 662.547 Mbytes per second
Loop count 6..... ok 662.533 Mbytes per second
Loop count 7..... ok 662.578 Mbytes per second
Loop count 8..... ok 662.601 Mbytes per second
Loop count 9..... ok 662.566 Mbytes per second
Loop count 10..... ok 662.469 Mbytes per second
root@zoom: #

Figure 4-5, memTest running random block test of onchip core-specific memory for 32 cores, with performance measurement enabled

Note that in the above command, the `-W` parameter indicates "write only". Also `-R` can be entered for read-only. If neither is entered, then the test assumes both read and write.

Also note in the above command that "performance measurement" has been specified using the --p option; in this case, specifically, the –pt option, which measures "transfer time only", excluding any intermediate software functions such as data preparation, error checking, etc.

Figure 4-6 below is a screen capture of a successful random block test of onchip shared memory (i.e. MCSM, or multicore shared memory) for 32 cores, using SigC667x hardware, by executing the following command:

```
#./memTest -cSIGC66XX -B0x100000 -pt -P -r -m0xffffffff -Mos -n10
```

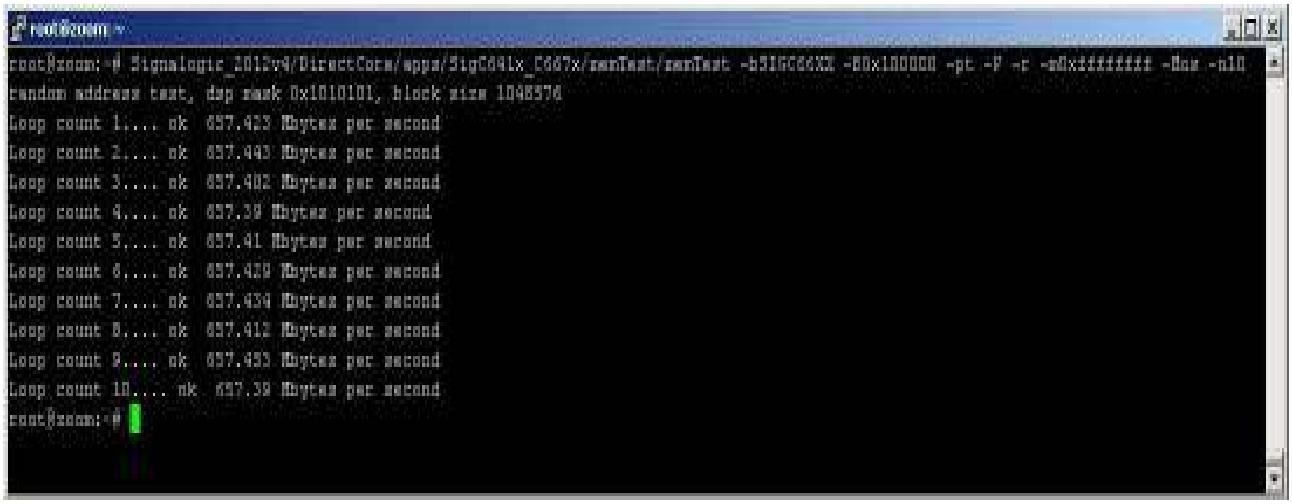
root@zoom: ~
root@zoom: # Signalogic_2012v4/DirectCore/apps/SigC641x_C667x/memTest/memTest -bSIGC66XX -B0x100000 -pt -P -r -m0xffffffff -Mos -n10
random address test, dap mask 0x1010101, block size 1048576
Loop count 1.... ok 657.423 Mbytes per second
Loop count 2.... ok 657.443 Mbytes per second
Loop count 3.... ok 657.402 Mbytes per second
Loop count 4.... ok 657.39 Mbytes per second
Loop count 5.... ok 657.41 Mbytes per second
Loop count 6.... ok 657.429 Mbytes per second
Loop count 7.... ok 657.434 Mbytes per second
Loop count 8.... ok 657.412 Mbytes per second
Loop count 9.... ok 657.433 Mbytes per second
Loop count 10.... ok 657.39 Mbytes per second
root@zoom: #

Figure 4-6, memTest running random block test of onchip shared memory for 32 cores, with performance measurement enabled

 Copyright © Signalogic, Inc. 2012 - 2016

Note in the above command, if the memory area specified by the memTest command is shared between all cores of a multicore CPU or target device (e.g. MCSM memory or external DDR3 SDRAM memory), then the memory area is actually tested only once. In this case, memTest will report test results only for the first core of multiple cores per device, which can be seen in Figure 4-6 above by examining the "mask" value printed along with test results.

4.1.2 Block Size Parameter Notes

1. For legacy SigC641x hardware, 4 kByte is the maximum block size for all memory areas. For SigC667x, 1 MByte is the maximum block size for external memory and shared memory (MCSM), and 96 kByte is the maximum size for core-specific L2 memory.

2. Up to a point, larger block sizes will increase measured performance; however, improvement is marginal beyond some point. Smaller block sizes are likely to decrease performance. Using SigC667x as an example, from 1 MByte down to 100 KByte block size, performance may decrease about 40 MByte/sec, at 50 KByte block size performance may decrease another 35 MByte/sec, and at 10 KByte block size performance may decrease as much as 200 MByte/sec and be as low as 350 MByte/sec.

3. If a SigC641x module is being used with a PCIe adapter containing the IDT (formerly Tundra Semiconductor) TSi384 PCIe-toPCI bridge chip, then a carefully selected block size should be specified in memTest commands, for example:

```
#./memTest –m0xa8 –r –B204 –n20 –cSIGC64XX
```

The `–B204` parameter specifies the block size at 204 bytes. This is the maximum block size for burst reads/writes that can be used without errors occurring when using a PCIe adapter with the Tundra chip. Specifying this block size compensates for an apparent limitation on PCIe transfer throughput with the TSi384 bridge chip.

4.2 boardTest

***Note:** this section assumes (i) the DirectCore target multicore CPU driver has been built and is currently loaded (either during the automated install process, or manually as explained in section 3.2 above, Building DirectCore Drivers, and (ii) DirectCore libraries have been built (either during the automated install process, or manually, as explained in section 3.3 above, Building DirectCore Libraries).*

The boardTest program gives an example of communicating with target code as it runs in real-time, demonstrating DirectCore library API calls.

To build the boardTest program, enter the following commands:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/boardTest

#make
```

Before running boardTest, first enter:

```
#./boardTest -h
```

This will bring up the help menu in order to configure SigC667x hardware.

Command line option syntax (options with "+" sign are mandatory):

-f<*num*>	+	Device clock rate in MHz (e.g. -*f1000*)
-m<*num*>	+	Core select bit mask (e.g. -*m1*, means core0, -*m2* means core1). Entry can be in hex, for example -m0x3f, which means run using 6 cores
-e<*string*>	+	Target executable file, for example -*ebdtest.out*. (ELF and COFF formats are supported for target executable files)
-c<*string*>	+	Card designator, for example -*cSIGC66XX*-8, which indicates that 8 cores should be allocated, or -*cSIGC66XX-32*, which indicates that 32 cores should be allcoated
-T		Run with talker enabled
-v		Run in verbose mode
-h		Display this help list

Here is an example boardTest command line:

```
# ./boardTest -m1 -f1250 -ebdtest.out -cSIGC66XX-8
```

Note that in the above command-line, the "`-ebdtest.out`" option specifies a target CPU executable code file (in ELF or COFF format). The clock rate will be 1 GHz or 1.25 GHz as specified by the `-f1000` or `-f1250` command line parameters (note that the `-f` command line parameter uses MHz as its units).

4.2.1 boardTest Output Description

Figure 4-7 below is a screen capture of output from the boardTest test program, executed using the following command for a SigC667x PCIe card:

```
# ./boardTest -m1 -f1250 -ebdtest.out -cSIGC66XX
```

The screen capture in Figure 4-7 shows a sequence of low frequency sinusoidal signal values. The values are read by the boardTest program from dual real-time data buffers in target memory and displayed repeatedly (about 1 sec rate), and will change in each read cycle. Figure 4-7 shows about four (4) of the read cycles -- note the changing data pattern.

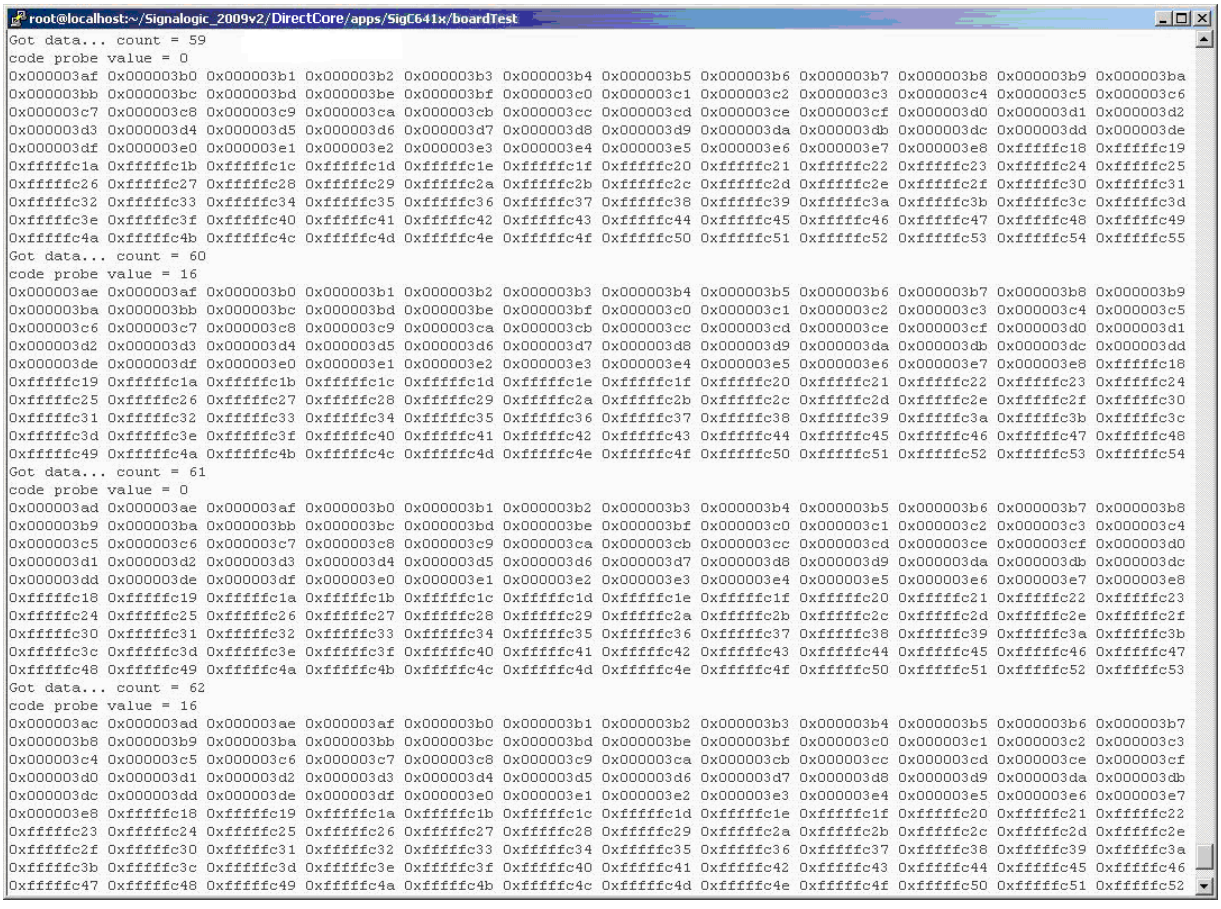

Figure 4-7, Example boardTest output display

To exit from the boardTest program, press the q (Quit) or ESC keys.

Note: ***you can also exit using Ctrl-C, but using the interactive keyboard commands is preferred as that allows the test program to free (de-allocate) the card handle, close its instance of the driver, and perform other clean up operations. This helps to maintain "clean" host Linux / server operation, and avoid memory leaks, driver errors, etc***

4.2.2 C++ Target

It's also possible to build boardTest by specifying a C++ target, which causes a different source code file to be used during the make. The C++ source code uses a "target class" to reduce the number of library calls and somewhat simplifies the amount of code necessary to set up basic target card / module connectivity and data transfer. To build with the C++ target, enter the following *(comments in italics)*:

```
#make cpp    (uses C++ source code)
```

Currently, the "straight make" (no target specified) is the standard (default) option, and should always be tested first.

Note: ***The cpp target is not well maintained and may give some build warnings or errors. Please contact Signalogic if you will need to write application programs using C++***

4.3 fftTest

***Note 1:** This section assumes you have already (i) built and loaded the DirectCore target multicore CPU driver, as explained in section 3.2 above, Building DirectCore Drivers, and(ii) built all DirectCore libraries, as explained in section 3.3 above, Building DirectCore Libraries.*

***Note 2:** Currently fftTest is only supported by SigC667x hardware.*

The fftTest program gives an example of running a parallelized algorithm on multiple target cores, using DirectCore library API calls.

To build the fftTest program, enter the following commands:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/fftTest

#make
```

Before running fftTest, first enter:

```
#./fftTest -h
```

This will bring up the help menu in order to configure SigC667x hardware.

Command line option syntax (options with "+" sign are mandatory):

-f<num>	+	Device clock rate in MHz (e.g. -f1000)
-m<num>	+	Core select bit mask (e.g. -m1, means core0, -m2 means core1.(Note: in this test, only one bitmask values of 2^n-1 can be selected, specifying the number of cores to run in parallel)
-e<string>	+	Target executable file (in ELF or COFF format, e.g. -efft.out)
-c<string>	+	Card designator (e.g. -cSIGC66XX)
-a<num>		Algorithm flag, 0 = parallel FFT, 1 = serial FFT (default is parallel)
-n<num>		FFT order, from 2 to 20 (default = 6)
-I<num>		Input waveform type, 0=ramp, 1 = Dirac Delta pulse
-i<string>		Input waveform file (in .wav or .tim format)

```
-v                  Run in verbose mode

-h                  Display this help list
```

Here is an example fftTest command lines:

```
# ./fftTest -m1 -f1250 -efft.out -cSIGC66XX -
```

Note that in the above command-line, the "`-efft.out`" option specifies a target CPU executable code file (in ELF or COFF format). The clock rate will be 1 GHz or 1.25 GHz as specified by the `-f1000` or `-f1250` command line parameters (note that the `-f` command line parameter uses MHz as its units). Parameters -a, -n, -I, and -i can be used to specify algorithm and input options.

4.3.1 fftTest Output Description

Figure 4-8 below is a screen capture of output from the fftTest test program, executed using the following command for a SigC667x PCIe card:

```
# ./fftTest -m1 -f1000 -efft.out -cSIGC66XX -n4
```

The screen capture in Figure 4-8 shows output for a small FFT order using a parallelized method based on Bailey's "Four Step" algorithm [1], which "factors" an FFT of order N into an NxM matrix of FFTs, where N and M are as close as possible to √N. FFTs in the matrix are then run on multiple target CPU cores concurrently.

A small FFT order can be used to allow results to conveniently viewed, and the test to be repeated in serial (non-parallelized) FFT mode in order to compare values for correctness.

[1] FFTs in External or Hierarchical Memory, David H. Bailey, 1989. Journal of Supercomputing, vol. 4 no. 1 Mar 1990. http://www.davidhbailey.com/dhbpapers/fftq.pdf

```
fftTest$ ./fftTest -m1 -f1000 -offt.out -cSIGC66XX -n4
DirectCore / CIM FFT test program for C64x / C66x hardware
userSpecified = {clock = 1000 MHz, Target Executable = fft.out, Card Type = SIGC66XX, Core List = 1, Algorithm Flag = -1}
CardStr = SIGC66XX
IObaseAddr = 0
Core clock rate (Hz) = 1000000000
Target executable file = fft.out
Algorithm mode flag -1
FFT mode: Parallel
Input type 0
FFT Order is 4
Initializing SigC667x card...
Initializing SigC667x cores with core list 0x1...
DSInitProcessor: qwCoreList = 0x1
SlotIndx = 0
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitBoardT50 success
DSXInitCard Successful!
Resetting specified devices and cores...
DSXResetCard
DSXResetCard Successful!
Downloading run-time code to specified cores...
Entry point = 0xc000000
test mode addr = 0x802094
Setting multicore CPU Properties...
dwCoreId_addr = 8020a4
writing sin/cos table data, addr = 0xc00d760, M=4, N=4
offset value = 32
Sincos Table
sin[0] = 1.000000        cos[0] = -0.000000
sin[1] = 0.000000        cos[1] = -1.000000
sin[2] = -1.000000       cos[2] = -0.000000
sin[3] = -0.000000       cos[3] = 1.000000
Sincos Table 2
sin[0] = 1.000000        cos[0] = -0.000000
sin[1] = 0.000000        cos[1] = -1.000000
sin[2] = -1.000000       cos[2] = -0.000000
sin[3] = -0.000000       cos[3] = 1.000000
writing test data, RI base addr = 0xc100080
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
inp[0]=0.00 0.00j       1.00 0.00j      2.00 0.00j      3.00 0.00j
inp[4]=4.00 0.00j       5.00 0.00j      6.00 0.00j      7.00 0.00j
inp[8]=8.00 0.00j       9.00 0.00j      10.00 0.00j     11.00 0.00j
inp[12]=12.00 0.00j     13.00 0.00j     14.00 0.00j     15.00 0.00j
Got data... count = 0
fft results:
out data[0] = 120.00 0.00j
out data[1] = -8.00 40.22j
out data[2] = -8.00 19.31j
out data[3] = -8.00 11.97j
out data[4] = -8.00 8.00j
out data[5] = -8.00 5.35j
out data[6] = -8.00 3.31j
out data[7] = -8.00 1.59j
out data[8] = -8.00 0.00j
out data[9] = -8.00 -1.59j
out data[10] = -8.00 -3.31j
out data[11] = -8.00 -5.35j
out data[12] = -8.00 -8.00j
out data[13] = -8.00 -11.97j
out data[14] = -8.00 -19.31j
out data[15] = -8.00 -40.22j
accelerator card runtime: 0.000008 sec
```

Figure 4-8, Example fftTest display output

4.4 streamTest

***Note:** This section assumes you have already (i) built and loaded the DirectCore target multicore CPU driver, as explained in section 3.2 above, Building DirectCore Drivers, and(ii) built all DirectCore libraries, as explained in section 3.3 above, Building DirectCore Libraries.*

The streamTest program gives an example of running a video codec algorithm on multiple target cores, using DirectCore library API calls.

To build the streamTest program, enter the following:

```
#cd /install_path/Signalogic/DirectCore/apps/SigC641x_C677x/streamTest

#make
```

Before running streamTest, first enter:

```
#./streamTest -h
```

This will bring up the help menu in order to configure SigC667x hardware.

Command line option syntax (options with "+" sign are mandatory):

-f<num>	+	Device clock rate in MHz (e.g. -f1000)
-m<num>	+	Core select list (e.g. -m1, = core0, -m2 = core1, -m3 = cores 1 and 2.(Note: in this test, only one core can be selected at a time)
-e<string>	+	Target executable file (in ELF or COFF format, e.g. -evideo.out)
-c<string>	+	Card designator (e.g. -cSIGC66XX)
-i<string>		Input unencoded video file (in .yuv format)
-o<string>		Output encoded video file (in .h264 format)
-x<num>		X dimension resolution
-y<num>		Y dimension resolution
-r<num>		Frame Rate (in frames per sec, default is 30 fps)
-B<num>		Bitrate of encoded (compressed) output file (default is 800 kbps)

```
-s<num>          +      Mode, -s0 = file, -s2 = streaming

-v                      Run in verbose mode

-h                      Display this help list
```

Here are some example streamTest command lines:

```
#./streamTest -m1 -f1250 -s0 -estream.out -cSIGC66XX-8
-i/install_path/Signalogic/video_files/airshow_352x288p_30fps_420fmt.yuv
-x352 -y288 -r30 -B1500000 -oencoded.h264
```

and

```
#./streamTest -m0x03 -f1250 -s2 -estream.out -cSIC66XX-8
-i/install path/Signalogic/video files/parkrun 720p 50fps 420fmt.yuv
-x1280 -y720 -r50 -B1500000 -oencoded.h264
```

Note that in the above command lines, the "`-estream.out`" option specifies a target CPU executable code file. Target CPU clock rates can range from 1 to 1.6 GHz as specified by the `-f` command line parameter (e.g. `-f1000, -f1250, -f1600`, etc. Note that the `-f` command line parameter uses MHz as its units). Parameters -i, -o, -x, -y, -r, and -s can be used to specify input and output files, resolution, frame rate, and mode. In `-s0` mode, streamTest operates in "one shot" file-to-file mode. In `-s2` mode, streamTest operates in continuous file-to-file or file-to-stream mode, where a stream is continuous network IP/UDP/RTP packet output.

Note: The High Performance Multicore Network (HPMN) stack software must be loaded on the target accelerator to connect and send RTP output streams (this software requires a license and may not be included in some eval/demo software packages).

4.4.1 streamTest Output Description

Figure 4-9 below is a screen capture of output from the streamTest program, executed using the following command for a SigC667x PCIe card:

```
#./streamTest -m1 -s0 -f1250 -estream.out -cSIGC66XX
-itouchdown_pass_1080p_30fps_420fmt.yuv -x1920 -y1080 -r30 -oencoded.h264
```

The screen capture in Figure 4-9 below shows typical streamTest run-time output, and Figure 4-10 below shows the resulting H.264 encoded output file viewed using the VideoLAN VLC player.

```
videoTest$ ./videoTest -m1 -f1000 -evideo.out -cSIGC66XX -itouchdown_pass_1080p_tr.yuv
DirectCore H.264 algorithm test program for C64x / C66x hardware
userSpecified = {clock = 1000 MHz, Target Executable = video.out, Card Type = SIGC66XX, Core List = 1, Algorithm Flag = -1}
CardStr = SIGC66XX
Core clock rate (Hz) = 1000000000
Target executable file = video.out
Initializing  SigC667x card...
Initializing SigC667x cores with core list 0x1...
DSInitProcessor: qwCoreList = 0x1
SlotIndx = 0
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitBoardT50 success
DSXInitCard Successful!
Resetting specified devices and cores...
DSXResetCard
DSXResetCard Successful!
Downloading run-time code to specified cores...
call to DSLoadFileCores Success
loading file: touchdown_pass_1080p_tr.yuv
DSLoadDataFile loaded 653184000 bytes from input video file touchdown_pass_1080p_tr.yuv
Entry point = 0xc000000
testrun symbol address = 0x85ce34
errorCode symbol address = 0x857310
numFramesEncoded symbol address = 0x857304
Host test run = 0
Target error code = 0
test mode addr = 0x802094
Setting multicore CPU Properties...
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
Timer running...
Num frames encoded:     210
C66x video encoding complete
Target test run = 101
Target error code = 0
Total number of frames encoded = 210
Number of output bytes85c478
Saving 9041343 bytes to output file encoded.h264
DSXResetCard
DSXResetCard Successful!

Stopped Hardware...
```

Figure 4-9, Example streamTest display output

As a progress update, streamTest will continuously update the current frame (after the line “Num frames encoded” in Figure 4-9 above). If streamTest hangs for any reason, you can exit by pressing the q (Quit) or ESC keys.

> ***Note:*** ***you can also use Ctrl-C to exit test programs, but the interactive keyboard commands are preferred, as they allow the test program to free (de-allocate) the card handle, close its instance of the driver, and perform other clean up operations. This helps maintain "clean" host Linux operation, and avoid memory leaks, driver errors, etc.***

Figure 4-10 below shows playback of the resulting “encoded.h264” file using the VideoLAN VLC or MPlayer video players (other widely used video players can also be used, such as Windows Media Player).

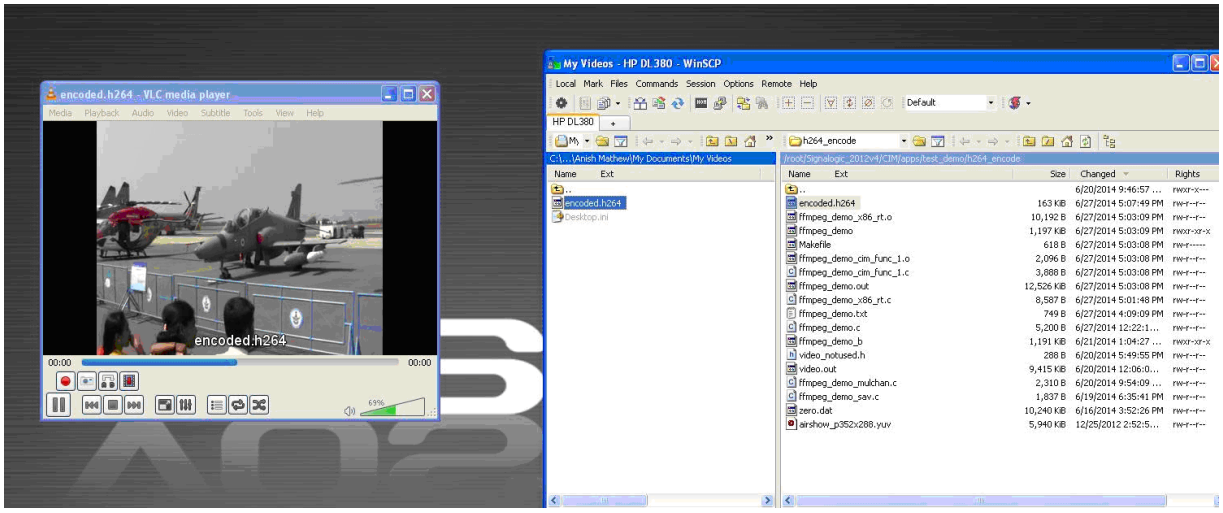

Figure 4-10, Example of encoded streamTest output. In the above screen capture, H.264 encoded data has been stored to a file, and then played back for test / comparison purposes using the VideoLAN VLC open source video player

Figure 4-11 below shows playback on a Ubuntu X11 desktop of the original .yuv file using ffplay, which is a component of ffmpeg. ffplay is available for Linux and Windows.

Figure 4-11, Example of original (unencoded) video input .yuv file data, viewed with ffplay

The following .yuv files should be installed under the folder

```
install_path/Signalogic/video_files
```

and can be used with the streamTest demo to provide some examples of compression quality at different resolutions.

Filename	Resolution	Frame Rate	YUV Pixel Format	Comments
airshow_352x288p_30fps_420fmt.yuv	352 x 288	30	4:2:0 [1]	Very short (less than 5 sec) video with intermediate encoding challenge. The camera moves so video images are changing frame-to-frame, but the background is of medium complexity
touchdown_pass_1080p_30fps_420fmt.yuv	1920 x 1080	30	4:2:0	About 19 sec video clip with sports action (football game)
touchdown_pass_640x480p_30fps_420fmt.yuv	640 x 480	30	4:2:0	Same, but 640x480 resolution
parkrun_720p_50fps_420fmt.yuv	1280 x 720	50	4:2:0	About 11 sec video clip of a man running in a park. This video provides a difficult encoding challenge as (i) the background is complex (forest and park scenery) and (ii) the camera moves so the video images are changing frame-to-frame
hallway_352x288p_30fps_420fmt.yuv	352 x 288	30	4:2:0	Two people in a hallway carrying and picking up objects. This video is used for image analytics moving object tracking demos
Akiyo_176x144p_30fps_420fmt.yuv	176 x 144	30	4:2:0	News anchor reading the news. This video may be used for some debug and test purposes

[1] 1.5 bytes/pixel

Table 4-1, streamTest example test videos

4.5 Setup for Video Streaming and Image Analytics Demos

Figure 4-12 below shows the basic setup for video and image analytics demos.

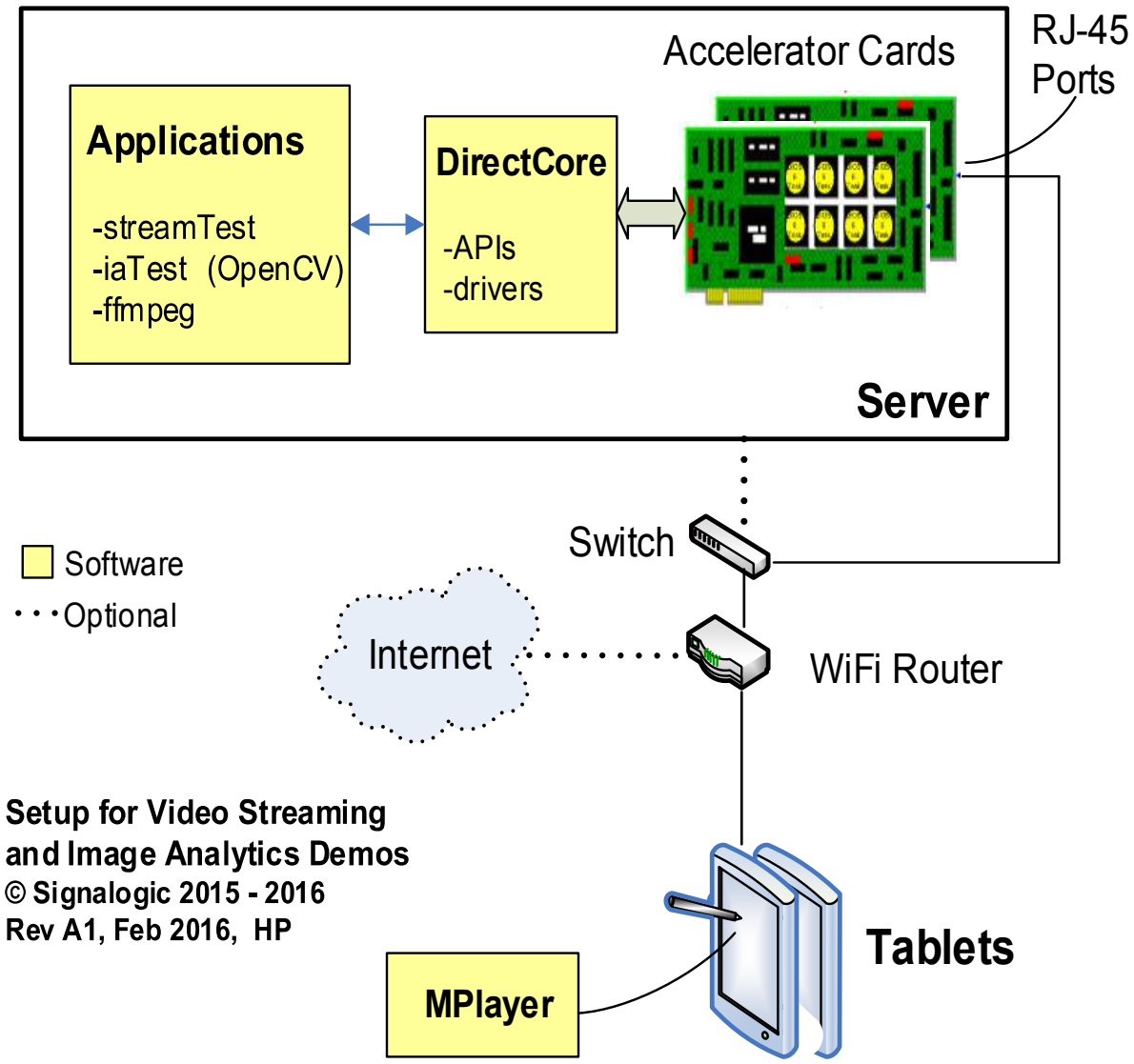

Figure 4-12, Setup for video streaming and Image Analytics Demos

Note that a modified MPlayer video player should be installed on tablets in use for test and demo (see section 4.5.1.1 below, Installing MPlayer). VLC, FF Player, VPlayer, and other players may work, but may exhibit high latency or other undesirable behavior. In general, off-the-shelf Android players are intended for entertainment purposes, with real-time communication and low latency as secondary objectives.

As an example of scaling up the configuration shown in Figure 4-12 above, Figure 4-13 below shows 16 tablets concurrently displaying real-time image analytics video streams from one SigC677x card (32 c66x cores, or two cores per stream). [1]

[1] The picture in Figure 4-13 was taken at booth 3323 at the Supercomputing 2015 exhibit in Austin, Texas.

Figure 4-13, Sixteen Android tablets concurrently displaying image analytics video streams from one SigC667x card

4.5.1 Tablet Configuration

Tablets used for video and image analytics demos should meet the following requirements:

- 720p resolution or higher (default demo configuration is 720p; demos can be configured for 1080p if supported by the tablet)
- Modified MPlayer video player installed (see section 4.5.1.1 below, Installing MPlayer)

Table 4-2 below gives a list of tablets tested with SigC667x card video streaming output.

Tablet	Android Version	Comments
Samsung Galaxy Tab 10.1	KitKat 4.4.2	
Samsung Galaxy Note	TBD	
Nexus 9 by Google	Lollipop 5.0.2	
Nexus 10 by Google	Jelly Bean 4.2.2	MPlayer modified to fix getnameinfo() source code issue
Amazon Kindle Fire HD 7"	TBD	

Microsoft Surface Pro 3	N/A	Ubuntu 12.04 installed

Table 4-2, Tablets tested with SigC667x accelerator card output video streaming

4.5.1.1 Installing MPlayer

To install the performance-modified MPlayer, go to this Github page to download the APK and install on tablets in use for test and demo:

http://github.com/MPlayer-Signalogic

MPlayer is a widely used video player with underlying code based on ffmpeg (linked into the Android executable as libmplayer.so). A thread regarding development of the Android version of MPlayer is located here:

http://forum.xda-developers.com/showthread.php?t=890761

Note that development was active in 2010 and 2011, and since then Signalogic has licensed MPlayer-Android source codes and continued to make improvements.

4.5.2 Internet Configuration

Android tablet players require Internet access in order to download an "sdp file" that will allow an IP video source to begin streaming. A minimum sdp file is simple and short, for example a typical sdp file looks like the following:

```
c=IN IP4
m=video 45056 RTP/AVP 96
a=rtpmap:96 H264/90000
```

Internet access can be provided by either (i) connecting a broadband input to the WiFi router, as shown in Figure 4-12 above, or (ii) configuring a web server on the test/demo server. Steps required for router based Internet access are given in section 4.5.3 below, and steps required for server based Internet access are as follows:

1. Enter command

```
yum install httpd
```

2. Enter commands

```
systemctl start httpd.service
systemctl enable httpd.service
```

4. Enter commands

```
firewall-cmd --permanent --zone=public --add-service=http
```

```
firewall-cmd --permanent --zone=public --add-service=https
firewall-cmd --reload
```

4. Store the sdp file at location /var/www/html/sig.sdp

5. Access the sdp file from the MPlayer url GUI (see section 4.5.4 below, Running MPlayer) to play the stream:

```
http://<IP_Address_of_Server>/sig/sig.sdp
```

4.5.3 Router Configuration

WiFi routers used for video and image analytics demos should meet the following requirements:

- 2.4 or 5 GHz
- 802.11n

If Internet access is connected to the router, then the following basic instructions for router configuration apply.

1. Power up the router and connect any wireless or wired device with it and open up a browser and give URL:

   ```
   http://<IP_ADDRESS_OF_ROUTER>
   ```

   Depending on how your LAN is configured, example might entries might be

   ```
   http://10.0.1.1
   http://192.168.0.1
   ```

   Once accessed, the router will ask for credentials to log in. After logging in, a screen similar to Figure 4-14a below should display.

Figure 4-14a, Router configuration main screen, after logging in

2. Select “Internet” or similar option (in Figure 14-4a above, the Internet option is at upper left, underneath “Home”). A screen similar to Figure 14-4b below should display.

Figure 4-14b, Router Internet Setup screen

3. Locate the “Internet IP Address” (or similar) configuration area. It should show options for:

   Dynamic IP address (obtain from ISP)
   Static IP address

   Select option 1 (Dynamic IP address). Press “Apply” (or “Save”, or similar depending on

the router type and make) to cause changes to take effect. A screen similar to Figure 4-14c below should be displayed.

Figure 4-14c, Router "updating settings" screen

4. After several seconds (possibly as long as a minute or two), the settings update should complete. Select the Home option (or "Main" or similar). Internet access should show as enabled.

4.5.4 Running MPlayer

Steps required to run the performance-modified version of MPlayer on tablet(s) in use for demo and test with SigC667x accelerator cards are as follows.

1. Click on the MPlayer-hpc icon, as shown in Figure 4-15a below.

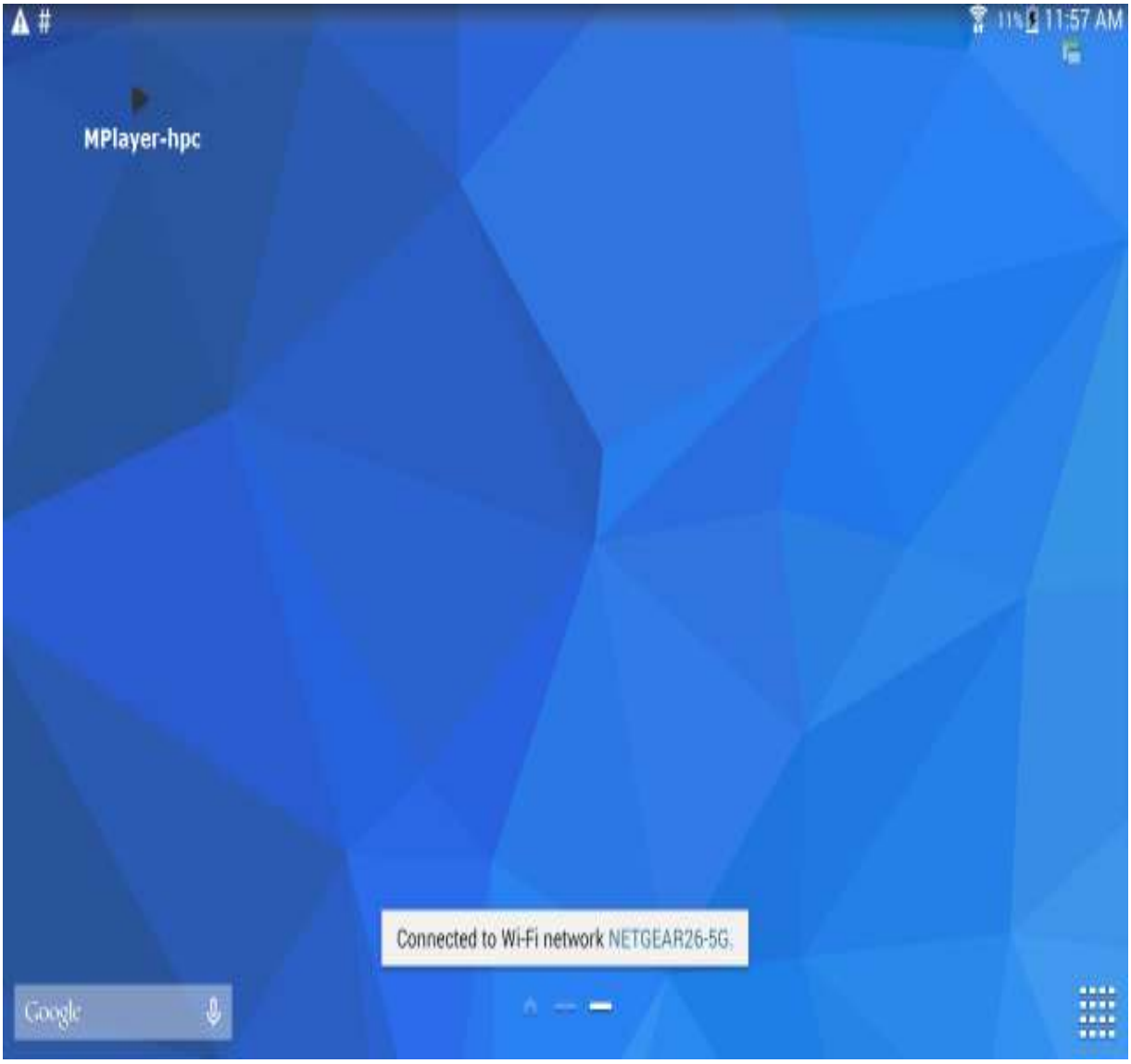

Figure 4-15a, Opening MPlayer-hpc

2. After opening MPlayer-hpc, the screen shown in Figure 4-15b below should appear.

 Copyright © Signalogic, Inc. 2012 - 2016

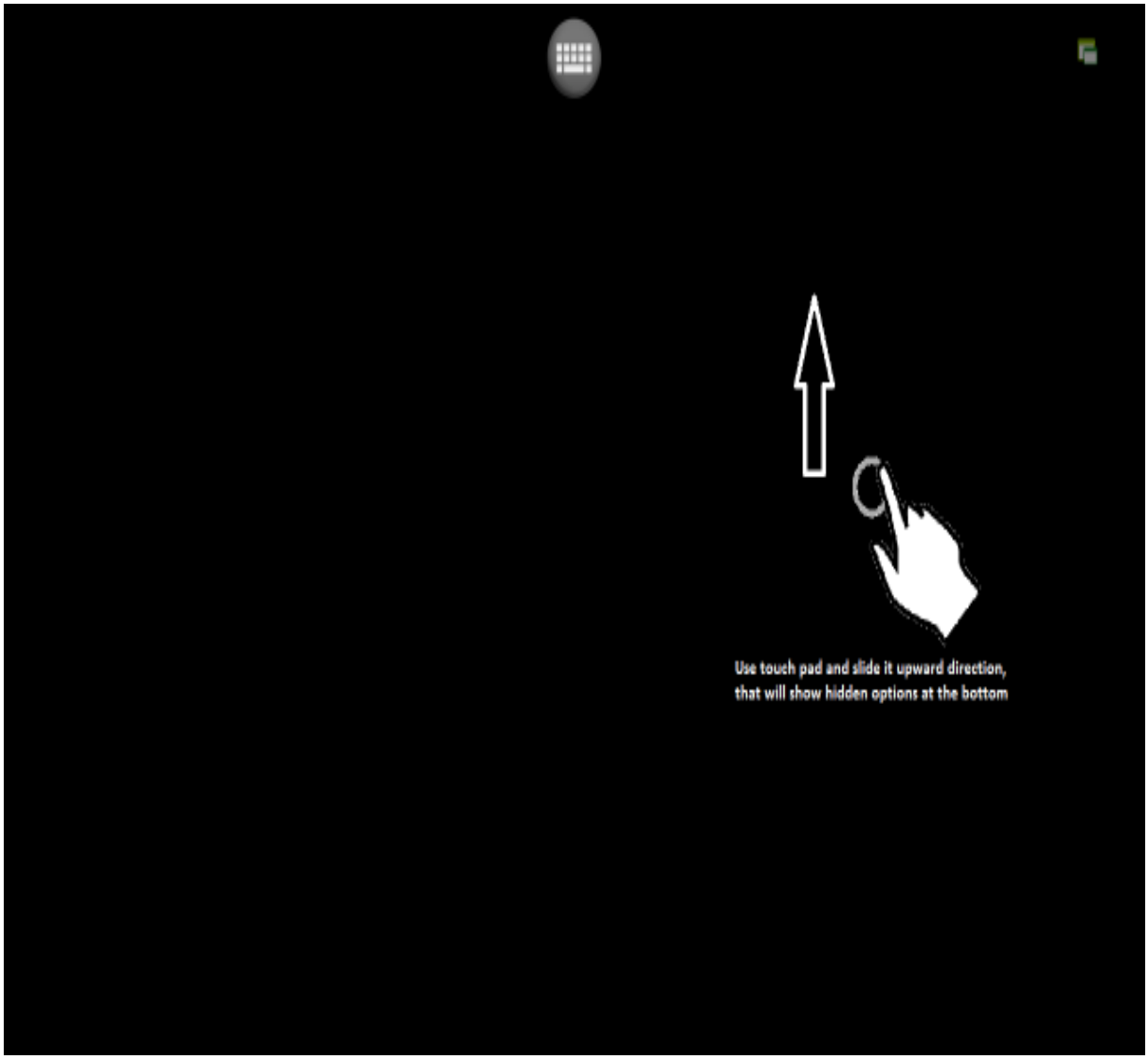

Figure 4-15b, MPlayer-hpc touch pad screen

3. Slide the touch pad upward to show the MPlayer-hpc settings icon, visible at lower right in Figure 4-15c below.

 Copyright © Signalogic, Inc. 2012 - 2016

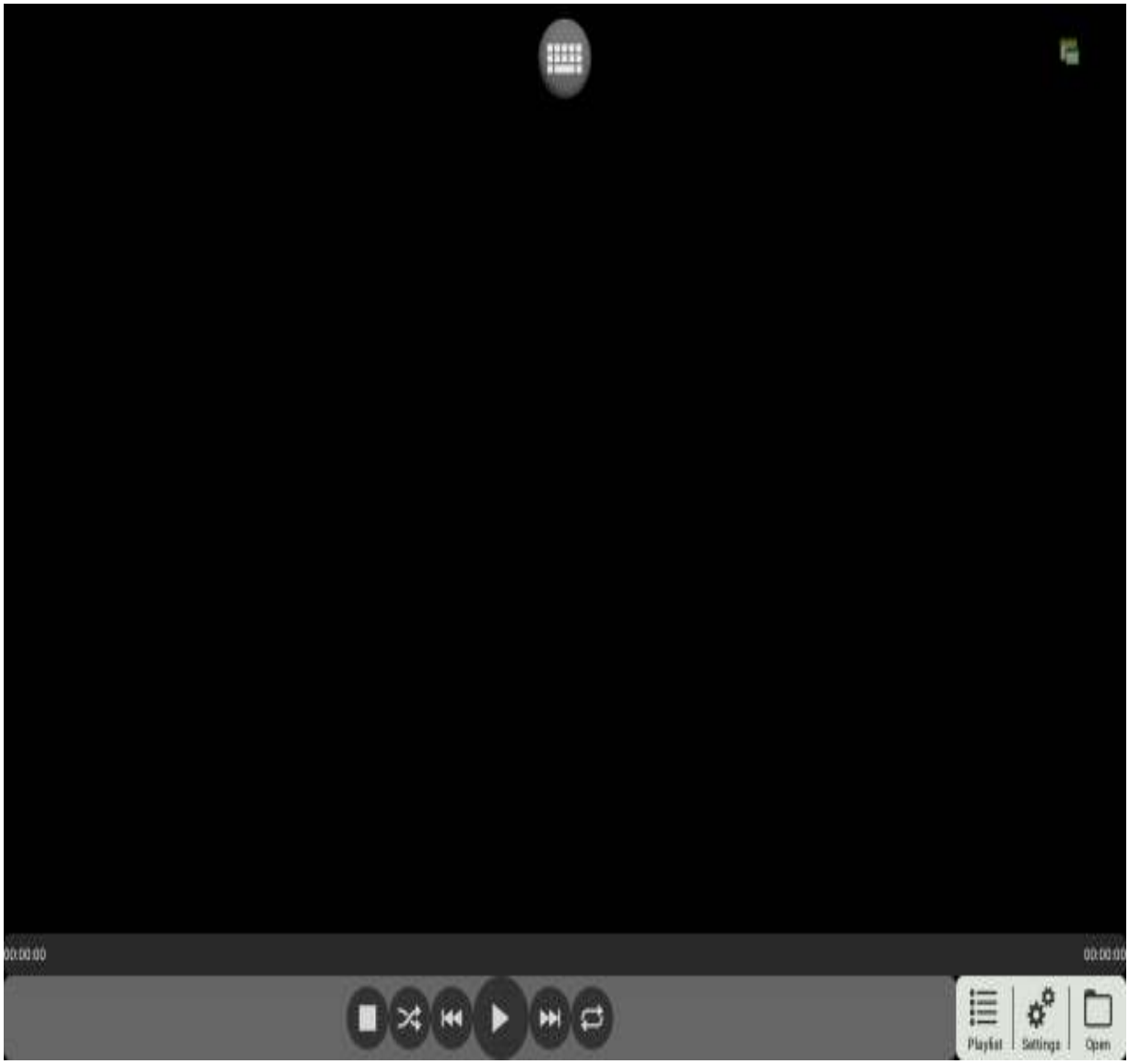

Figure 4-15c, MPlayer-hpc play window, with settings icon at lower right

4. Press "Settings" to bring up the menu in Figure 4-15d below.

 Copyright © Signalogic, Inc. 2012 - 2016

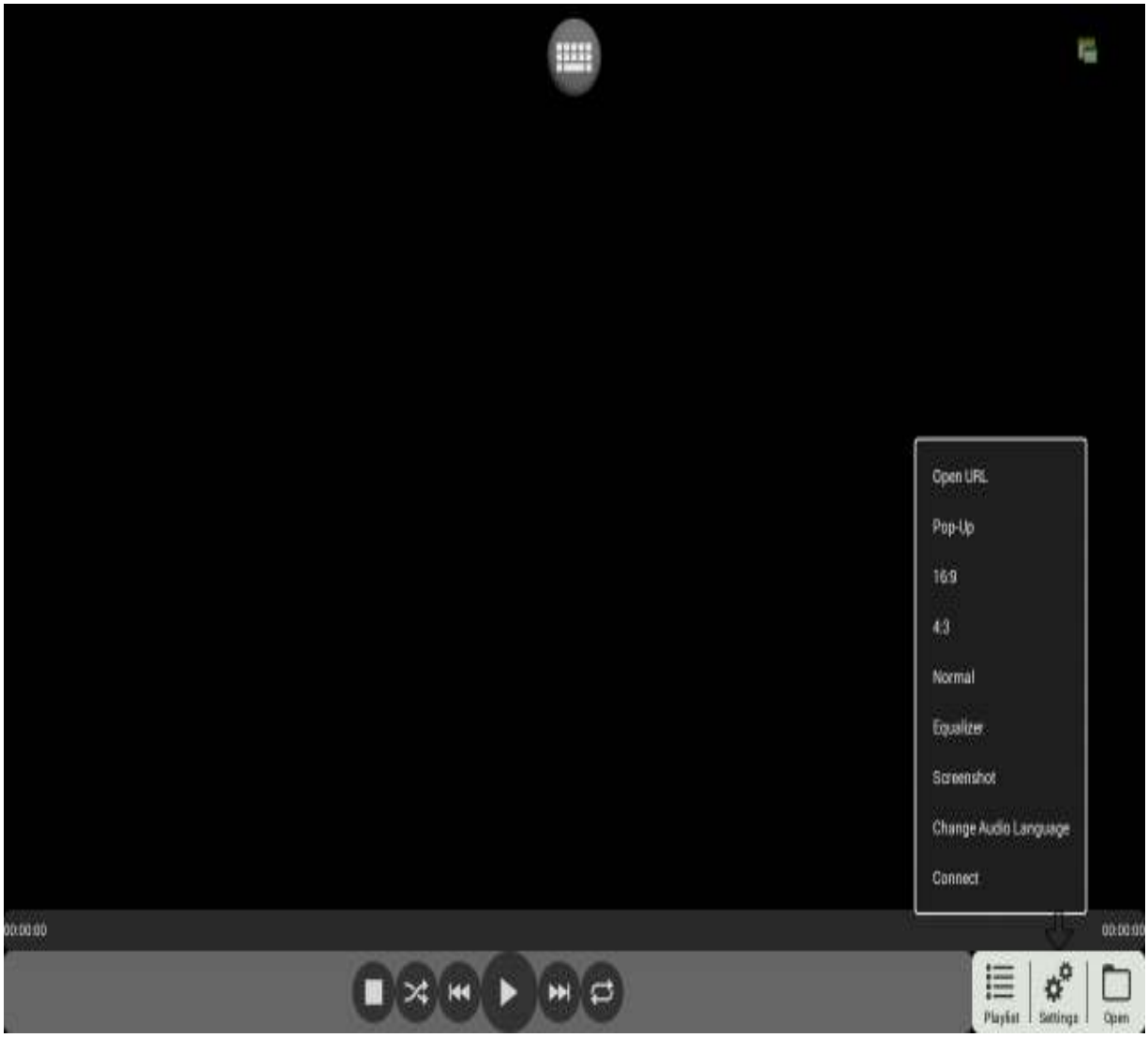

Figure 4-15d, MPlayer-hpc settings menu

5. Select the "Open URL" menu option to display an SDP file URL entry window, as shown in Figure 4-15e below.

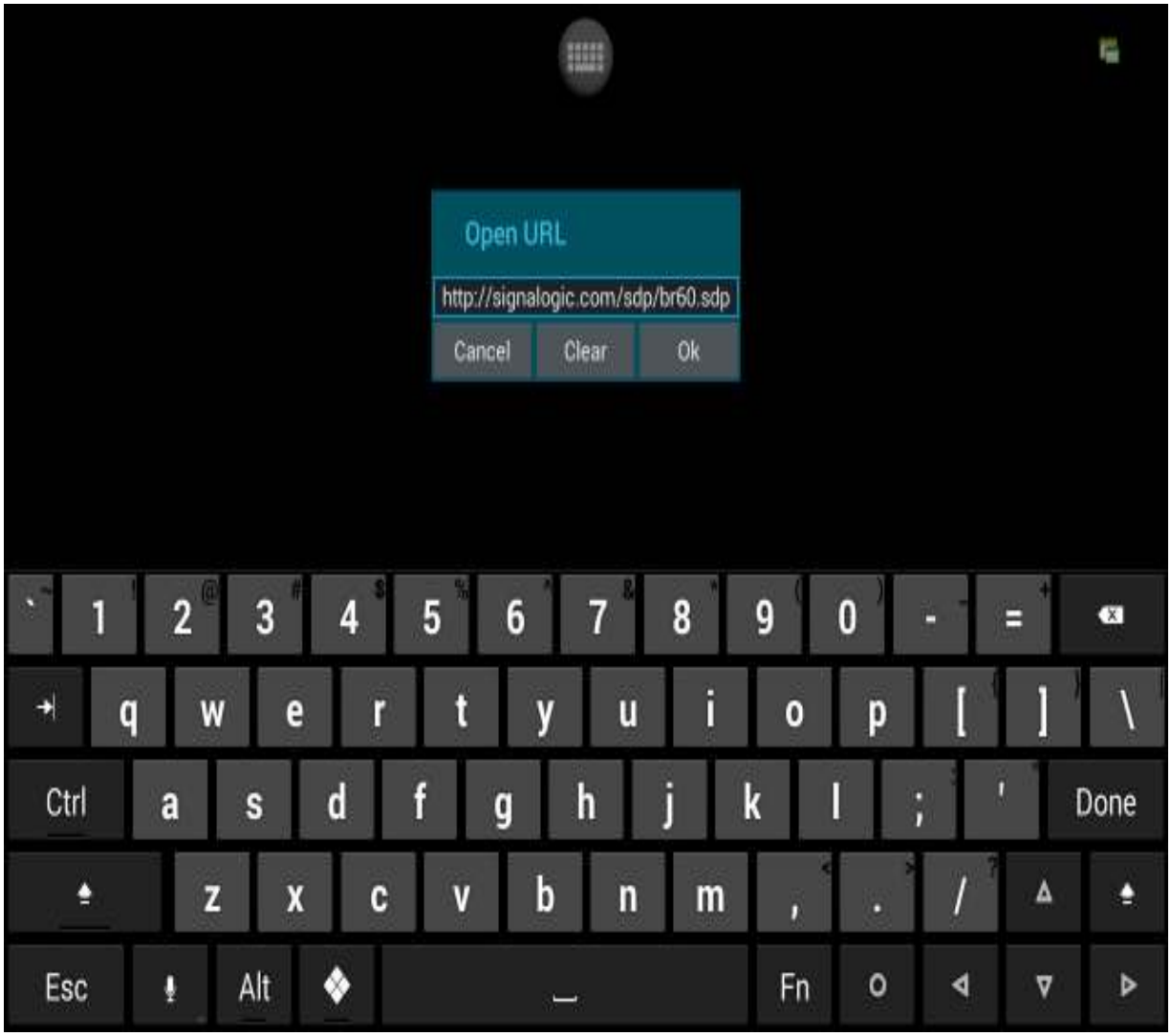

Figure 4-15e, MPlayer-hpc SDP file URL entry window

Note that a default SDP file is stored at http://signalogic.com/sdp/br60.sdp, but can be changed as needed.

6. Click "Ok" to start streaming.

 Copyright © Signalogic, Inc. 2012 - 2016

4.6 Setup for Transcoding Demos

**** THIS SECTION UNDER WORK *****

 Copyright © Signalogic, Inc. 2012 - 2016

5 VM Operation

***Note 1:** This section assumes one or more VMs installed in a Linux system configured with KVM + QEMU. VM installation is not covered here.*

***Note 2:** Screen captures in this section are taken from Ubuntu VMs. Other VMs may look different.*

All test programs and application demos described in section 4 can be run from VM (Virtual Machines) instances as well as host instances. In the case of a VM, the basic sequence looks like this:

- Configure SigC667x accelerator resources for the VM
- Run the VM
- If DirectCore software (and CIM and RTAF software, if applicable) have not yet been installed on the VM, then run the Automated Software procedure described in section 3.1 above, Installation Steps. This will (i) install Signalogic software and (ii) load required "front-end" (virtual I/O) drivers

  ***Note 3:** When running the automated software installation script, be sure to select "VM" when prompted !*

- Run test programs and application demos as required

5.1 VM Add and Configure Hardware

Steps required to configure a VM for SigC667x accelerator hardware resources are as follows.

1. First make sure the VM to be configured is shut off. One way to do that is the virsh (virtual shell) command:

```
virsh list –all
```

2. From the host Linux command line, run the Virtual Machine Manager (VMM) as shown in Figure 5-16a below.

root@localhost:/Harshal/Signalogic/DirectCore/VMM/virt-manager-1.1.0
File Edit View Search Terminal Help
[root@localhost ~]# cd /Harshal/Signalogic/DirectCore/VMM/virt-manager-1.1.0/
[root@localhost virt-manager-1.1.0]# ll
total 244
-rw-r--r--. 1 root root 734 Apr 16 2014 autobuild.sh
-rw-r--r--. 1 root root 17987 Jul 9 2012 COPYING
drwxr-xr-x. 4 root root 4096 Mar 3 18:53 data
-rw-r--r--. 1 root root 2103 Jul 28 2014 HACKING
-rw-r--r--. 1 root root 534 Apr 16 2014 INSTALL
drwxr-xr-x. 2 root root 4096 Nov 10 17:41 man
-rw-r--r--. 1 root root 474 Apr 16 2014 MANIFEST.in
-rw-r--r--. 1 root root 17267 Sep 7 2014 NEWS
-rw-r--r--. 1 root root 226 Sep 7 2014 PKG-INFO
drwxr-xr-x. 2 root root 4096 Nov 10 17:41 po
-rw-r--r--. 1 root root 1335 Jul 28 2014 README
-rw-r--r--. 1 root root 20732 Sep 6 2014 setup.py
drwxr-xr-x. 12 root root 4096 Nov 10 17:40 tests
drwxr-xr-x. 2 root root 4096 Nov 10 17:41 ui
drwxr-xr-x. 2 root root 4096 Mar 3 18:47 virtcli
-rw-r--r--. 1 root root 7582 Jul 28 2014 virt-clone
drwxr-xr-x. 2 root root 67 Nov 10 17:41 virtconv
-rw-r--r--. 1 root root 4347 Jul 28 2014 virt-convert
drwxr-xr-x. 2 root root 4096 Mar 3 18:47 virtinst
-rw-r--r--. 1 root root 30918 Sep 7 2014 virt-install
-rwxrwxrwx. 1 root root 9543 Jul 28 2014 virt-manager
drwxr-xr-x. 2 root root 4096 Mar 3 18:47 virtManager
-rw-r--r--. 1 root root 20682 Sep 7 2014 virt-manager.spec
-rw-r--r--. 1 root root 20686 Sep 7 2014 virt-manager.spec.in
-rw-r--r--. 1 root root 15113 Jul 28 2014 virt-xml
[root@localhost virt-manager-1.1.0]# ./virt-manager
[root@localhost virt-manager-1.1.0]#

Figure 5-16a, Running virt-manager from host Linux command line, in order to display the VMM GUI

After running virt-manager, a screen similar to Figure 5-16b below should be displayed.

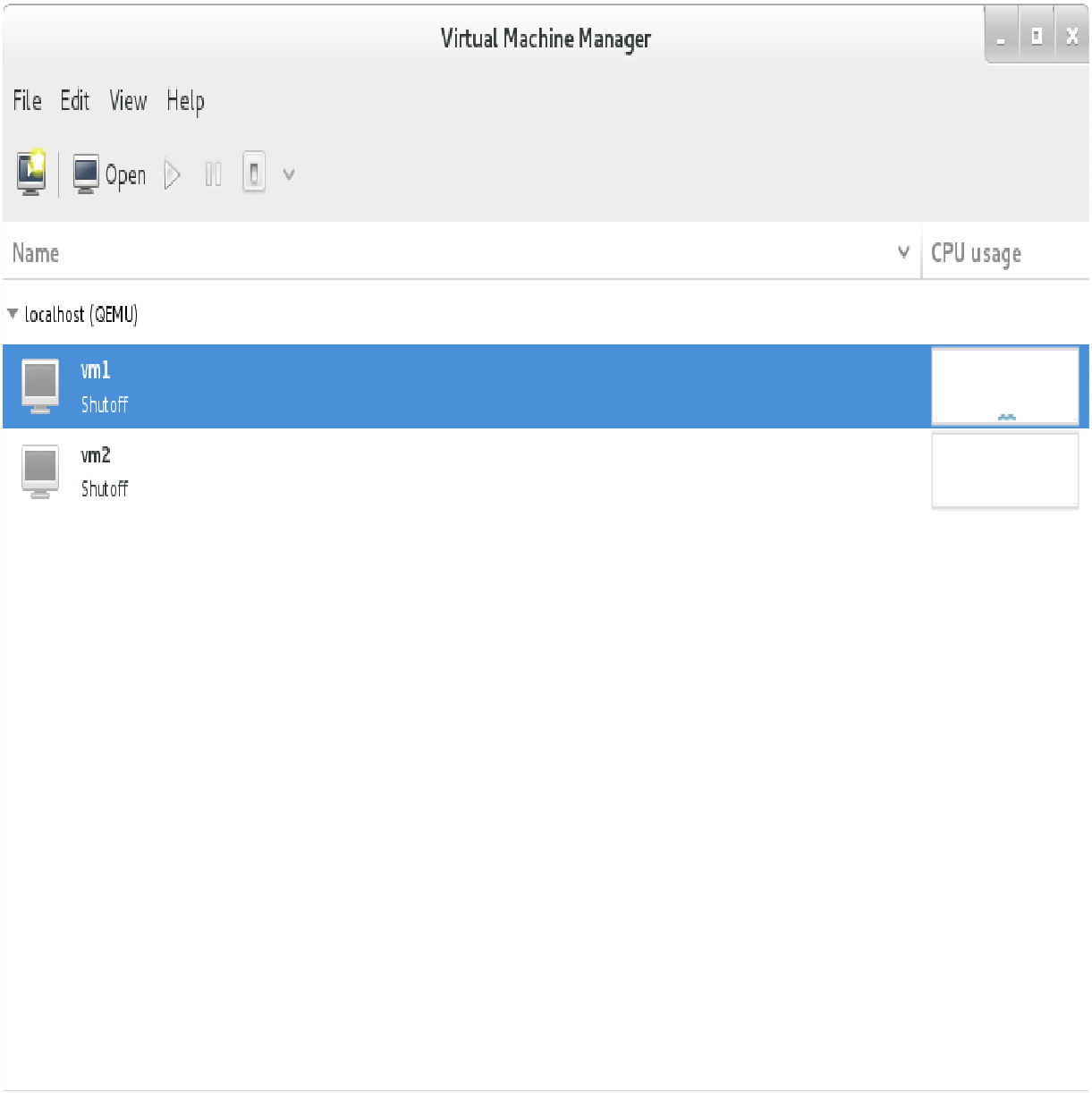

Figure 5-16b, VMM screen showing installed VMs and indicating whether they are shut off

3. Double click on the VM to be configured, which should display a screen similar to Figure 5-16c below.

 Copyright © Signalogic, Inc. 2012 - 2016

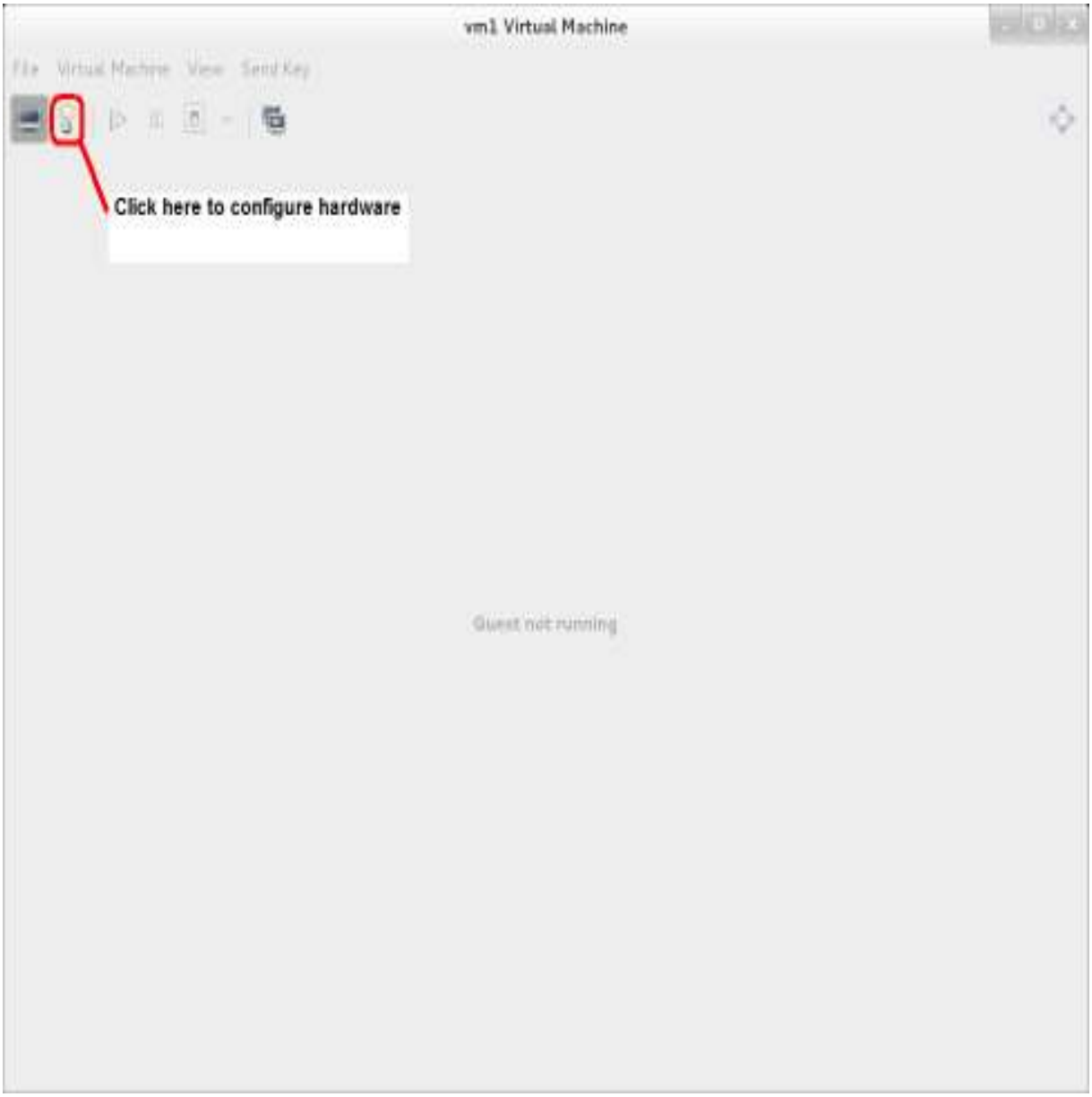

Figure 5-16c, VMM display for VM that is shut off ("guest not running")

Note 4: ***It's possible to add/configure SigC667x accelerator hardware in a running VM, but in that case the VM must be shut off and restarted for changes to take effect. This method is not recommended, but if you do this then a screen similar to Figure 5-16d below should appear.***

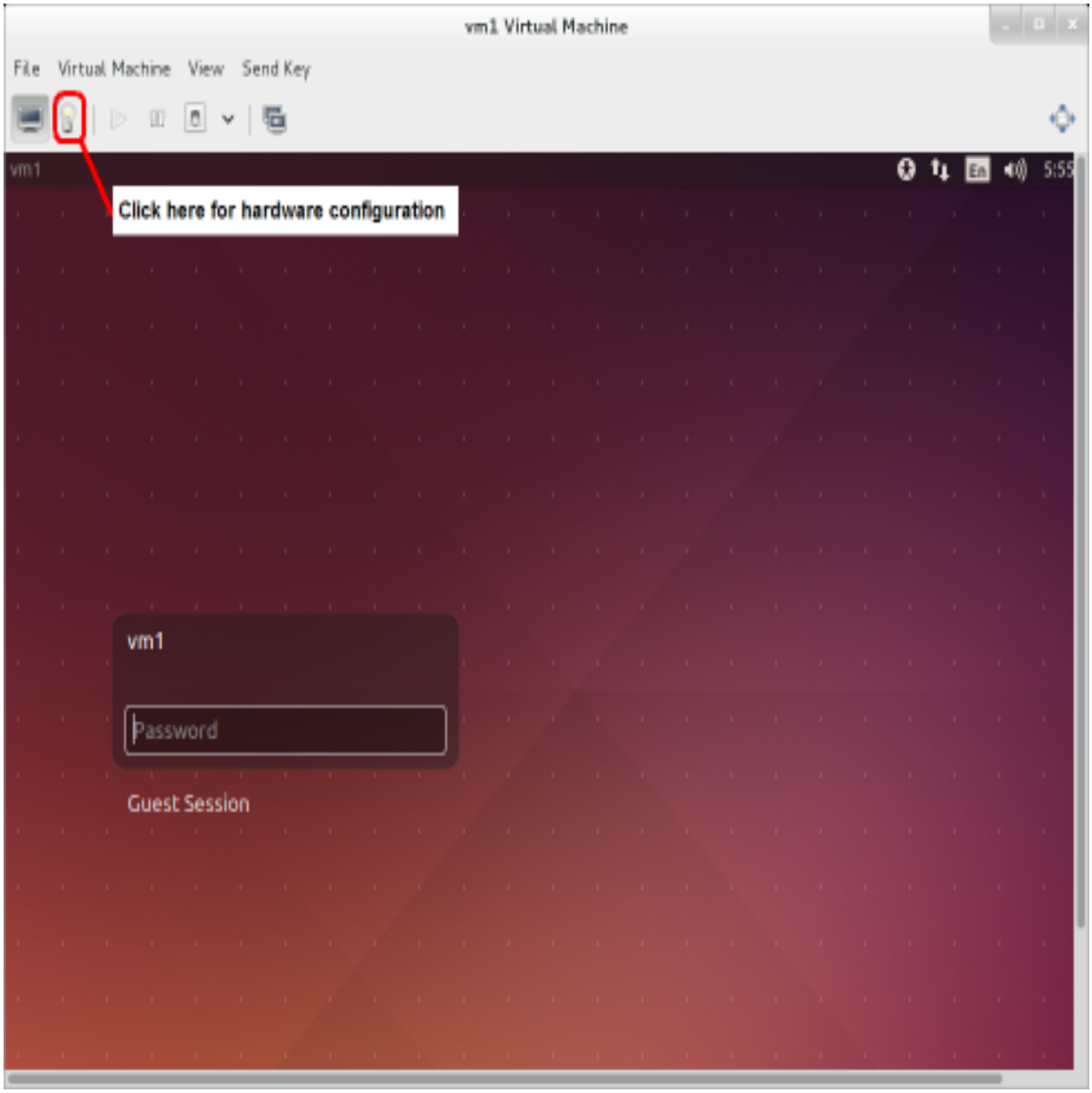

Figure 5-16d, VMM display for VM that is still running (this method not recommended)

4. In either case above, click on the "Details" icon shown in Figures 5-16c and 5-16d above (the Details icon looks like a lighbulb and has been circled in red in the figures). A VM configuration screen similar to Figure 5-16e below should then be displayed, including Hardware configuration, Basic.Details, and Hypervisor Details.

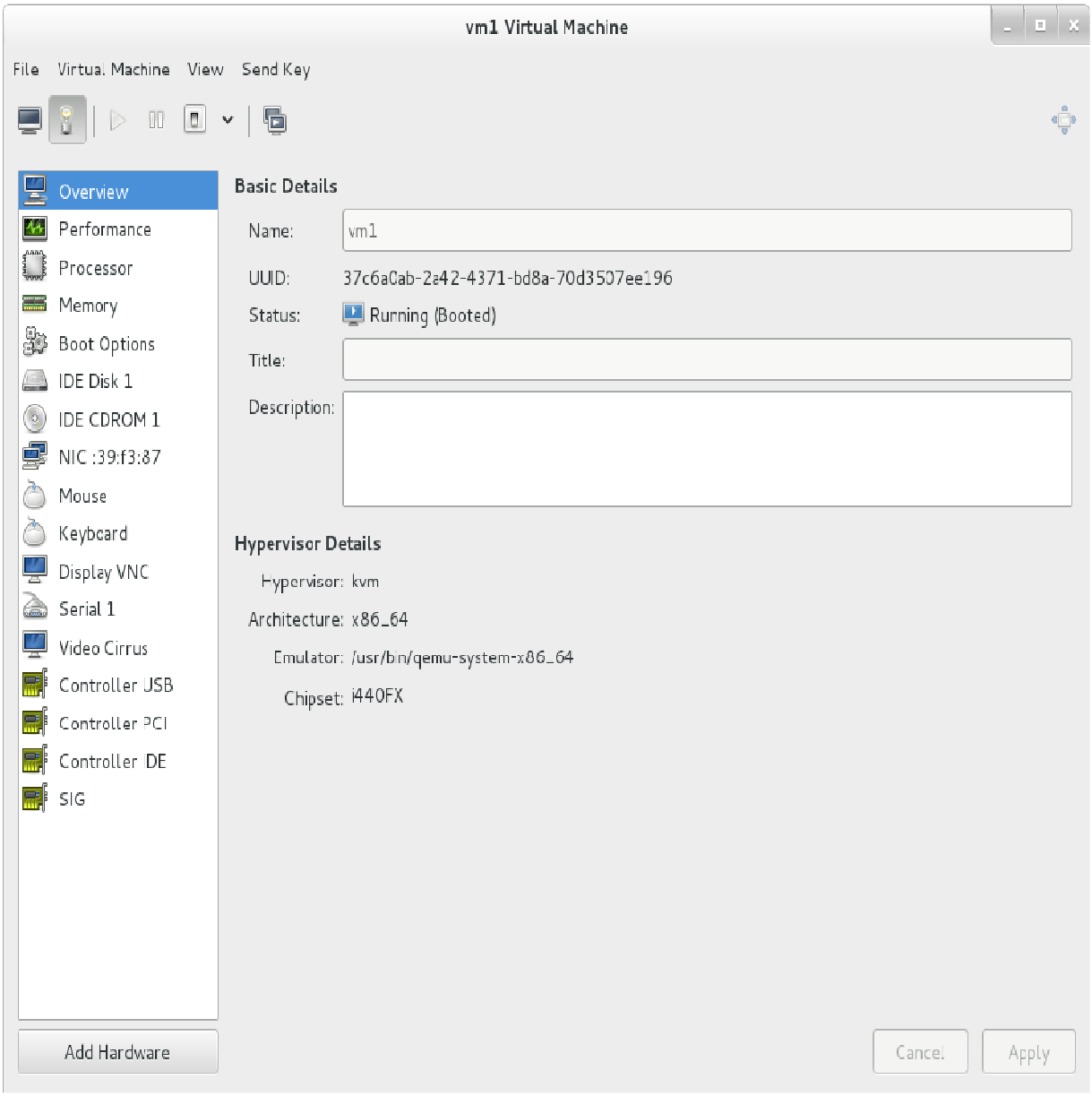

Figure 5-16e, VM configuration for Hardware Resources, Basic Details, and Hypervisor Details

Note the "SIG" hardware item in the hardware list at the left hand side (last item on the list) in Figure 5-16e above. If SigC667x accelerator hardware has not yet been configured, then SIG will not appear. After SigC667x hardware has been configured once, then subsequent displays of this screen will show SIG, and right-clicking on SIG will allow SigC667x hardware to be reconfigured as needed.

5. Click on "Add Hardware" as shown in Figure 5-16e above, which should bring up a display similar to Figure 5-16f below.

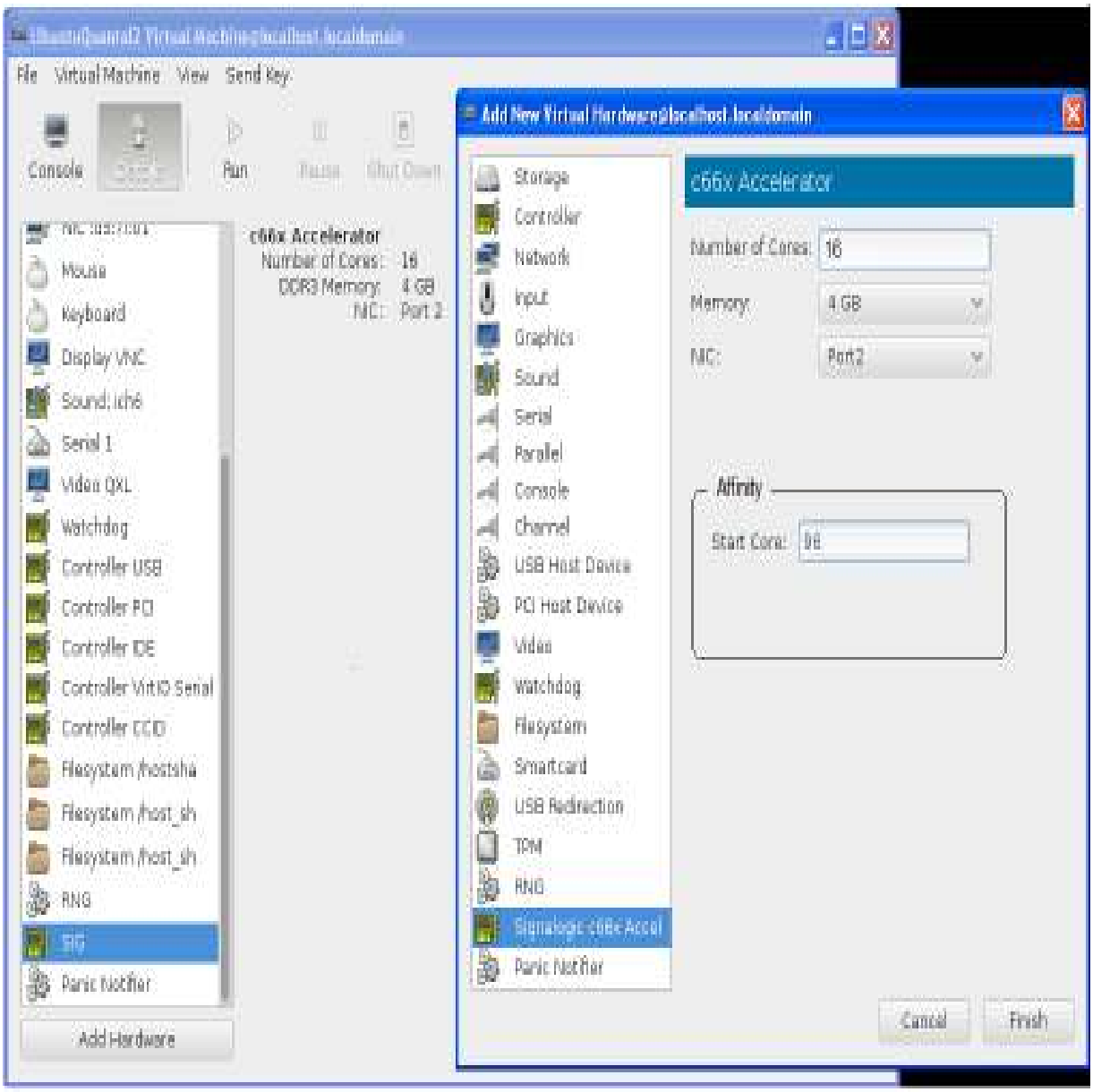

Figure 5-16f, Add Hardware options screen display

Figure 5-16f above shows "Add Virtual Hardware" dialog box options available when configuring a VM for SigC667x accelerator hardware. Note that the "SIG" label changes to read "Signalogic c66x Accel".

For detailed information and recommended settings when adding and configuring SigC667x accelerator hardware, see Section 5.2 below.

6. After clicking on Finish, the display should return to the screen shown in Figure 5-16e.

 Copyright © Signalogic, Inc. 2012 - 2016

Note that if the SIG device is being added for the first time, and the VM is shut off, then after clicking on Finish, the SIG device will appear in the hardware list after returning to the screen in Figure 5-16e. If however the VM is already running, then the SIG device will not appear (will not be "attached") until the VM is shut off and restarted.

Note 5: ***If for any reason drivers need to be manually loaded in a VM instance, follow the instructions in section 3.2.1 above, Loading a DirectCore Driver.***

 Copyright © Signalogic, Inc. 2012 - 2016

5.2 VM Hardware Resource Allocation

The following are guidelines for configuring SigC667x accelerator card resources. Please refer to Figure 5-16f above, "Add hardware options screen display".

- Cores should be allocated in groups of 8. Some examples:
  - the system has one (1) SigC667x-64 card installed. Eight (8) VMs allocate 8 cores each
  - the system has one (1) SigC667x-64 card installed. One VM allocates 16 cores and another VM allocates 48 cores
  - the system has two (2) SigC667x-64 cards installed. One VM allocates 32 cores, another VM allocates 64 cores, and a third VM allocates 32 cores

  In the future it may be possible to allocate cores per VM with finer granularity, but for now this is the supported procedure
- Memory is automatically assigned in chunks of 2 GB (2 GByte) per 8 cores
- Network I/O ports are automatically assigned in groups of 32 cores; i.e. cores 0-31 are on Port0, cores 32-63 on Port1, cores 64-95 on Port2, etc
- Currently the Affinity sub-dialog is not active. In the future the Affinity option will allow a starting core to be assigned, if for any reason specific physical c66x cores must be assigned to a VM

In the screen caps and notes above, "Memory" and "NIC" refer to DDR3 memory, 1 GbE Ethernet ports, and RJ-45 connectors located on SigC667x cards.

 Copyright © Signalogic, Inc. 2012 - 2016

5.3 VM Remove Hardware

Steps required to remove SigC667x accelerator hardware in a VM are as follows.

1. First make sure the VM to be configured is shut off, as noted in section 5.1 above, VM Add and Configure Hardware.

2. With the VMM Details interface displayed, as shown in Figure 5-16e above, "VM configuration for Hardware Resources, Basic Details, and Hypervisor Details", right-click on the "SIG" device at left hand side, and select "Remove Hardware". This sequence should look something like Figure 5-17a below.

 Copyright © Signalogic, Inc. 2012 - 2016

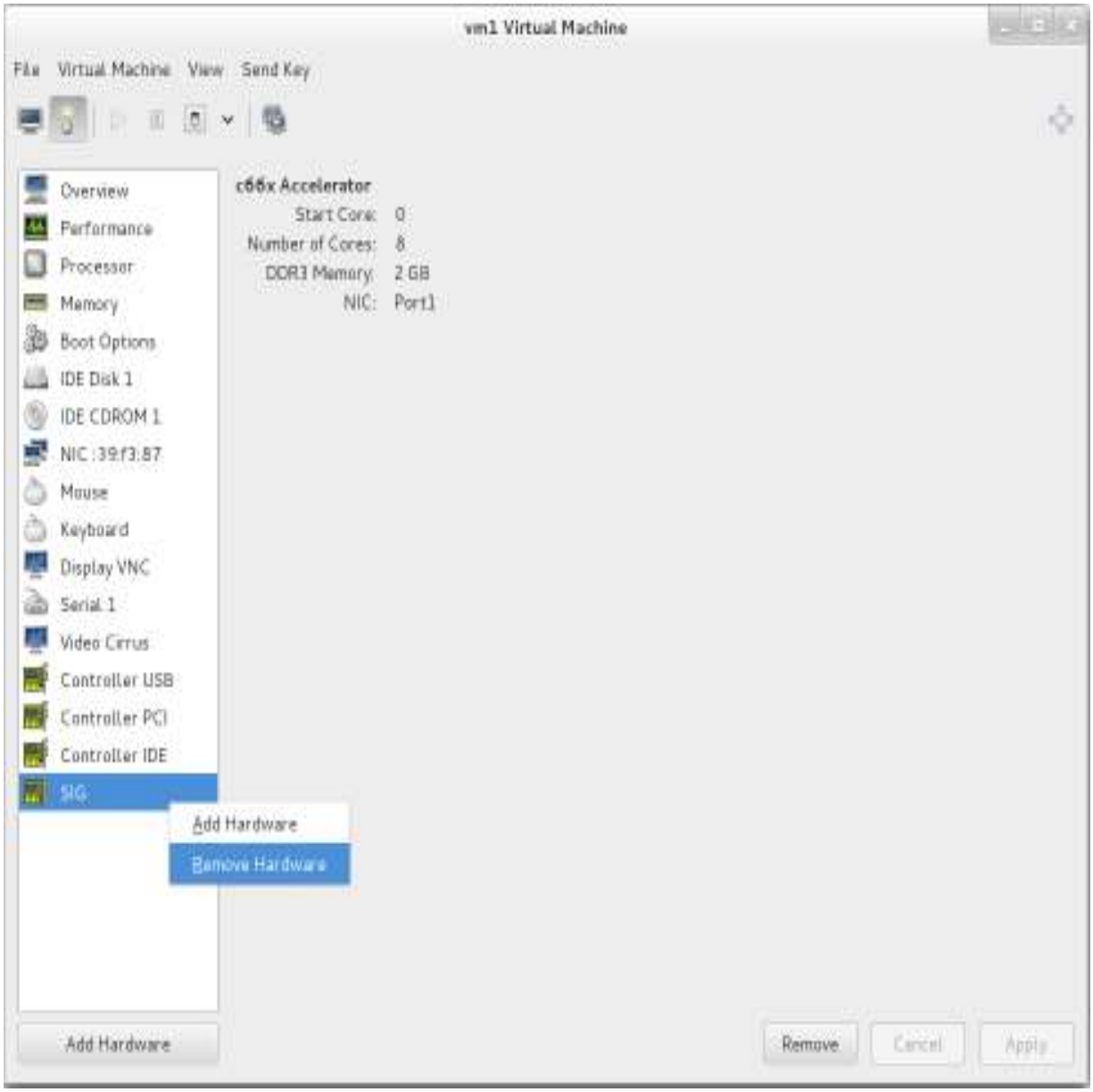

Figure 5-17a, Remove hardware

3. After selecting Remove Hardware, a screen with an “are you sure” prompt similar to figure 5-17b below should be displayed.

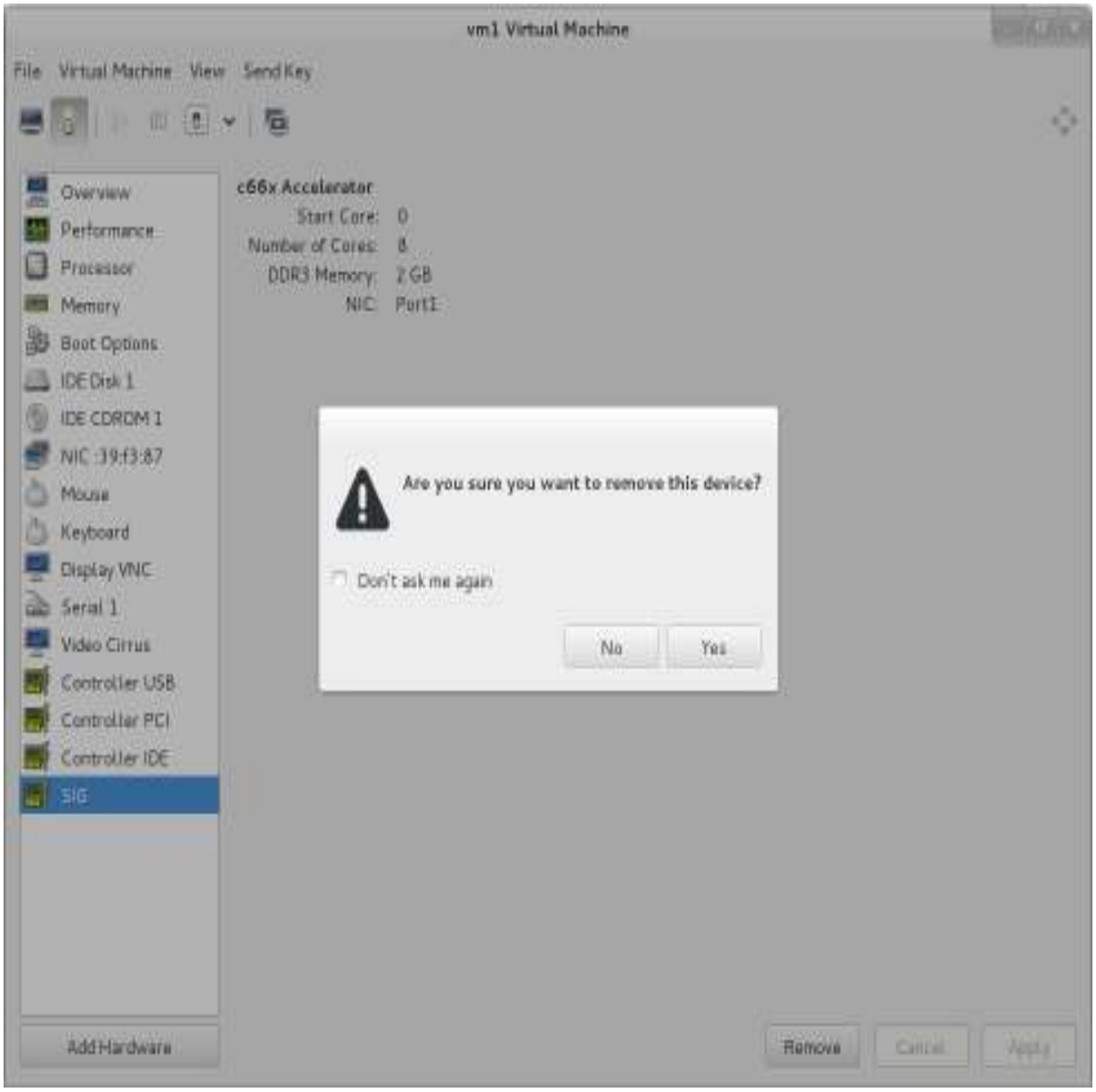

Figure 5-17b, Remove hardware prompt

4. If the VM is running then a screen to figure 5-17c below may be displayed.

 Copyright © Signalogic, Inc. 2012 - 2016

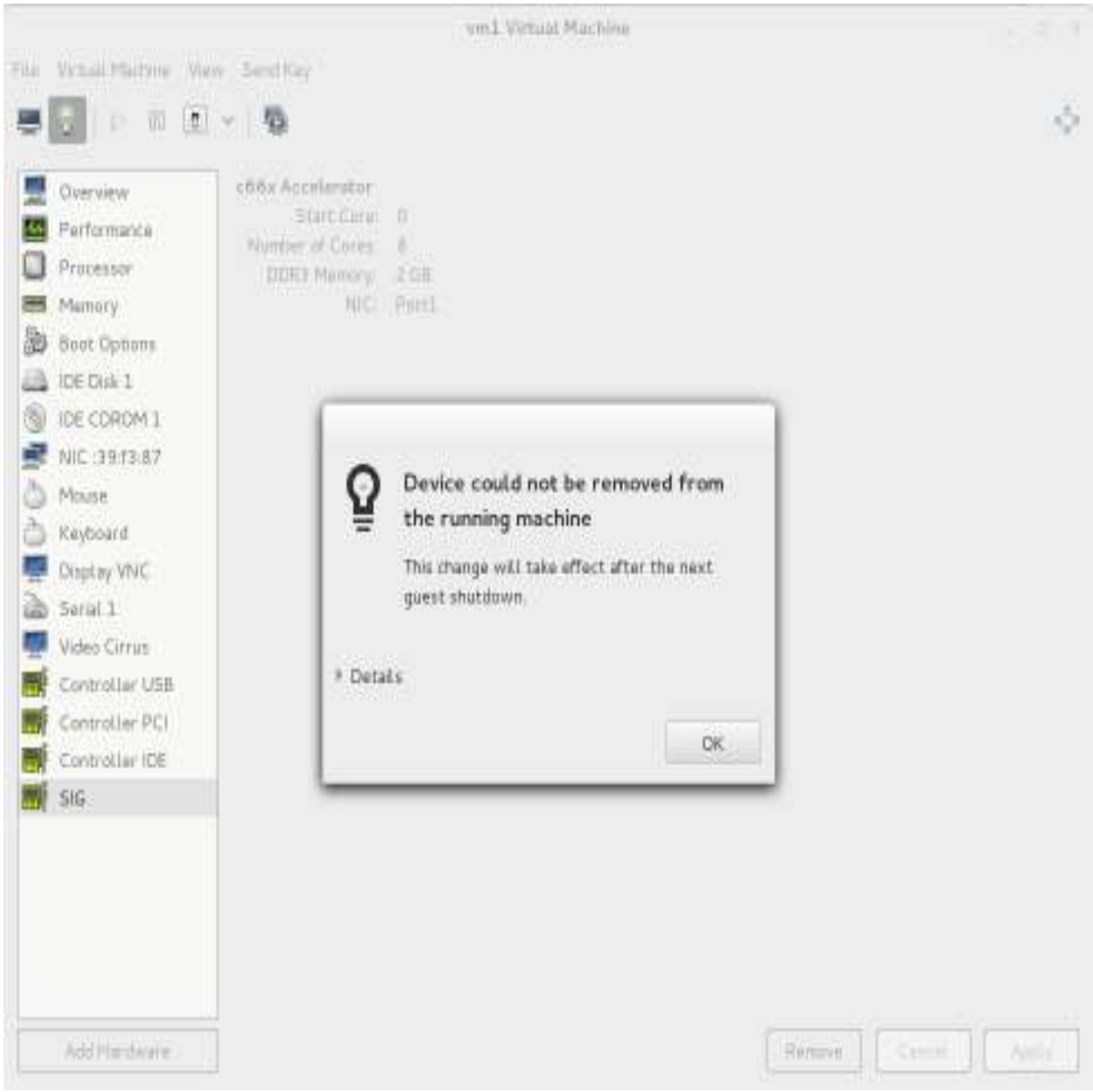

Figure 5-17c, Message indicating hardware removal won't take effect until the next shutdown

 Copyright © Signalogic, Inc. 2012 - 2016

6 DirectCore Software

***Note:** This section provides overall information about DirectCore Software. Detailed API and usage information is located in the DirectCore Reference Guide, located at ftp://ftp.signalogic.com/documentation/DirectCore/*

DirectCore software consists of the following main components, which are installed for all licenses (trial, demo, production, etc):

Component	Binary Files	Description
Libraries	hwlib.so, hwlib.a cimlib.so, cimlib.a hwmgr.a enmgr.a filelib.a tdmlib.a	User space libraries providing an API interface to user space application programs
Front-end driver	sig_virtio.ko	Kernel space VirtIO driver for VMs (Virtual Machines) that interfaces to QEMU (a KVM component)
Physical driver	sig_mc_hw.ko	Linux kernel space driver that interfaces physically to accelerator card hardware
Test programs and application demos	memTest boardTest fftTest streamTest mediaTest	Test programs, utilities, and application demos

Table 6-1, DirectCore software components

If you have a DirectCore software license, then source files will also be installed, as shown in Figure 3-2 above.

The DirectCore Software Reference Guide is located at

ftp://ftp.signalogic.com/documentation/DirectCore/

6.1 DirectCore Libraries

DirectCore libraries are user space shared objects (.so files) or static libraries (.a files) that provide an API interface to application programs. A general view of an application program sequence looks like:

assign	assign one or more card handles
init	init target CPU cores
reset	reset target CPU cores
load	download executable code to one or more target CPU cores
run	run one or more target CPU cores
communicate	read/write target core mem, mailbox communication, DMA-over-PCIe, etc
reset	
close	free card handles and other resources

The above is a simplified view; in practice it's typically more complex; for example, CIM applications, which are built automatically and create machine-generated application code, use advanced techniques. To see the above outline in action, the boardTest application demo is the DirectCore equivalent of a "Hello World" program (see section 4.2 above).

6.1.1 Card Handles and Core Lists

DirectCore APIs require a card handle. A card handle is typically assigned near the beginning of a C or C++ program, similar to the following:

```
hCard = DSAssignCard(NULL, szCardDesignator, nClockRate, 0, 0, 0, maxCores)
```

The card designator often times comes from the application program command line, although it can be defined and manipulated as needed as a string variable. For example, here is a typical command line for the boardTest demo program (see section 4.2 above), with a card designator of SIGC66XX-8:

```
./boardTest –m0xff –f1250 –ebdtest.out –cSIGC66XX-8
```

In this case, the card designator specifies a SigC66xx card and requests eight (8) cores from the system-wide pool of c66x cores. The other command line parameters in the above example are:

`-m0xff`	core list (8 cores in this example)
`-f1250`	clock rate of 1.25 GHz
`-ebdtest.out`	c66x executable code file bdtest.out

Most DirectCore APIs accept a core list parameter specifying up to 64 cores, and operate concurrently on specified cores. For systems with multiple cards, and thus core lists longer than 64 bits, additional card handles can be assigned.

 Copyright © Signalogic, Inc. 2012 - 2016

6.2 DirectCore Drivers

DirectCore drivers provide virtual and physical interfaces to accelerator card hardware and target CPU cores. DirectCore drivers support:

- A unified core list, allowing user space DirectCore APIs to operate on one or more cores simultaneously. The list is maintained across (i) card boundaries, allowing mixing and matching of cards and card types, and (ii) host and VM instances, allowing any combination of host and VM instances to run concurrently
- Thread-safe and process-safe concurrent access
- Asynchronous mailbox based transactions
- Various PCIe data transfer modes, including bus mastering, inbound transactions (DMA initiated by host software), and outbound transactions (DMA initiated by target CPU core software)
- Data sharing between host and target CPUs, or between target CPUs (basically a NUMA method of interprocessor communication)
- 32-bit and 64-bit BAR addressing modes. 32-bit is the default and recommended mode

6.2.1 Physical Driver

The DirectCore physical driver is composed of two files:

sig_mc_hw.ko	Host instance Loadable Kernel Module (LKM) file
sig_mc_hw.h	Header file, included in some test programs (e.g. memTest)

The header file contains detailed comments, but otherwise is not documented. The reason for this is that normally an application should not make calls directly to the DirectCore driver, for example, calls such as ioctl(), pread(), etc. There are a few exceptions to this, one being the DirectCore resource manager, or "Dashboard" program, that allows current core allocation, assignment, and usage to be displayed in real-time.

Driver source code is available under Source License Agreement (SLA).

6.2.2 Virtual Driver

The DirectCore physical driver is composed of two files:

sig_virtio.ko	Virtual Machine (VM) instance Loadable Kernel Module (LKM) file
sig_virtio.h	Header file

As with the DirectCore physical driver header file, the DirectCore virtual driver header file contains some comments and information, but otherwise is not documented.

 Copyright © Signalogic, Inc. 2012 - 2016

6.2.3 Driver Installation Notes

After software installation, DirectCore drivers will be automatically loaded at server boot time. To manually make, load, and unload DirectCore drivers, see section 3.2 above, “Building Direct Core Drivers”.

 Copyright © Signalogic, Inc. 2012 - 2016

7 RTAF Software

This section covers use of the Real-Time Algorithm Framework (RTAF) software. RTAF software includes the following components and run-time functionality:

- for TI targets, SYSBIOS configuration, including run-time initialization and memory allocation, timer configuration (OS tick, profile timers, etc)
- CSL, NDK, and other library and initialization
- Run-time initialization, including CPU boot
- Peripheral drivers and interrupt service routines (both hardware and software interrupts)
- Utility functions
- Algorithm functions
- DirectCore properties used for host communication with certain utility and algorithm functions
- CIM software support

7.1 RTAF Configuration Options

The RTAF software build process supports the following pre-processor definitions

Pre-Processor Definition	Description
`_USE_CM_`	Enables Call Manager support
`_PERIPH_TEST_`	Enables POST and other test options for peripherals
`_BIOS_` `_SYSBIOS_`	Enables SYSBIOS compatibility in the build process
`_CIMF`	Enables support required for CIM software builds
`_ANALOG_IO_`	Enables analog I/O support on SigC641x modules (default peripheral interface is McBSP0)
`_TDM_`	Enables TDM support on SigC641x modules (default peripheral interface is McBSP1)
`_SIGC66XX_`	Enables SigC66xx hardware support

_H110_	Enables H.110 support on SigC641x modules (default peripheral interface is McBSP2)
_USE_PN4_	Enables PN4 connector interface on SigC641x modules

Table 7-1, RTAF software pre-processor definitions

Pre-processor definitions can be set in the CCS project (under Project main menu item, Build Options, Compiler tab), or in Linux command-line build / make files.

7.2 RTAF Peripheral Support

RTAF supports the following C64x / C66x peripherals:

Peripheral	Comments
Host (HPI or PCIe)	HPI32 is supported on SigC641x, PCIe is supported on SigC667x
Ethernet (Network I/O)	If present on the hardware. In the case of SigC667x, the oncard 1 GbE interface is supported, using TI's NDK and PA (Packet Accelerator) interfaces
McBSP	For SigC641x, supports audio/analog I/O, TDM, H.110, and user-defined interfaces. Not currently supported on SigC667x (Telecom Serial Port, or TSIP, may be supported at a future point)
GPIO	Supports both SigC641x and SigC667x
Flash devices	Typically connected to EMIFB interface
FPGA Logic	If present on the card, for example host / CTBus FPGA on SigC641x modules
UART	If present on the hardware
RS-422	If present on the hardware

Table 7-2, RTAF software peripheral support

7.3 RTAF Development and Host Platform Notes

As noted in section 3 above, Software Installation, for SigC667x cards, RTAF software is installed under the mCPU_target folder on Linux systems as shown in Figure 7-1 below.

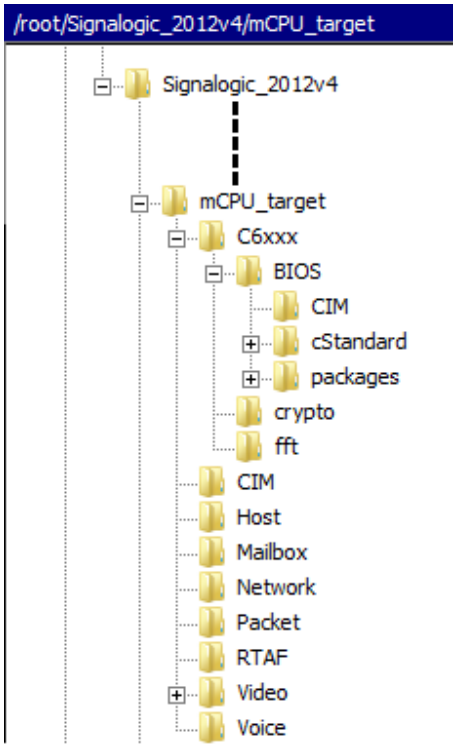

Figure 7-1, RTAF software folder hierarchy for SigC667x cards

There are a few fundamental points about RTAF development and host platform to keep in mind:

1. RTAF software supports SigC641x modules installed in Linux systems, but *SigC641x RTAF software is not generated or rebuilt on a Linux system*. Instead, the software must be maintained on a WinXP or Win7/8 system with Texas Instruments CCS software installed. Conversely, SigC667x RTAF software builds may be handled using CCS software or Texas Instruments command-line code generation tools (TI's Linux version of CGT). This is one reason why RTAF software installation looks different for C64x vs. C66x based hardware.

2. In Linux systems, RTAF software is stored on the "mCPU_target" subfolder. This naming convention refers to run-time software for multicore CPU or other target hardware; e.g. CIM accelerator, DSP card, video or telecom card, network processing card, neural net card, etc.

7.3.1 Linux Platform Builds

In Linux platform builds, there are two (2) important files:

- Makefile
- cimf_build.lst

The cimf_build.lst file is only used if CIM software is being used by host applications, and is located in:

```
/install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/
```

7.3.2 Building without CIM

1. First, go to the folder

   ```
   /install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/cStandard/
   ```

2. Then, use the 'make' command, which invokes the contents of the 'Makefile' file. Note that this file will be overwritten if/when a host-side CIM build script is used.

7.3.3 Building with CIM

To build with CIM host-side software use the CIM script 'cimpp.sh' located in:

```
/install_path/Signalogic/CIM/cimrt/
```

The command

```
./cimpp.sh -h
```

will display the command line options. An example build command might look like this:

```
./cimpp.sh -s/install path/Signalogic/CIM/apps/test/ffmpeg demo/ -px86
-ati66x -cSIGC66XX conv.c
```

The file 'template_Makefile' in that directory can be edited for changes to the build.

Makefile Notes:

1. ORDERED_OBJS includes any libraries (“lib files”) that need to be linked. More libraries can be added as needed.

2. CIM_DEFINES are all the preprocessor macro defines used.

3. CIM_INCLUDES are the include directories for the build process.

4. LINKER_INCLUDES are the include directories for the linker process.

There are 3 targets used in the makefile under the '# All Target' comment. The first one can be edited to change the platform used by changing the -p parameter. The second one builds the source files in the cimf_build.lst file. If CIM software is used by host applications then 2 additional source files will be added to the command to be built. The last tool is the linker, which uses ORDERED_OBJS and all the .obj files in the directory:

```
/install_path/Signalogic/mCPU_target/C6xxx/TI-RTOS/cStandard/
```

8 CIM® Hyperpiler™ Software

To install CIM Hyperpiler software unzip the sig_sw_host_cim_vNN.rar file. You should see the following

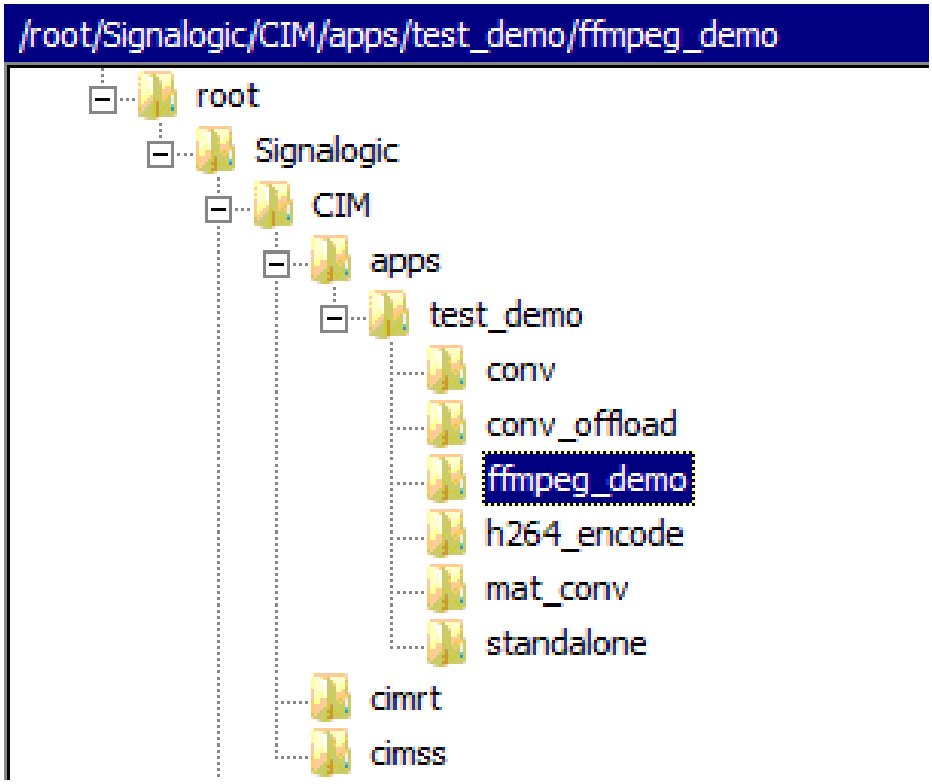

Figure 8-1, CIM host software folder hierarchy, after installation

8.1 Running User Programs with CIM

To run user programs with the CIM source code stream generation process, generally follow these guidelines

1. First modify the system PATH environment variable as follows:

```
export PATH=$PATH:/install_path/Signalogic/CIM/cimrt
```

This can be done each time the system boots, or made an automatic boot setting inside the shell profile configuration file, for example /etc/profile, etc.

2. cd (change directory) to the folder with your input C source files.

3. Enter a command line in the following form to run the CIM pre-processor:

```
cimpp <file1.c> <file2.c> <fileN.c> [-pPlatformType] [-aAccelType] [-cCardType]
```

 Copyright © Signalogic, Inc. 2012 - 2016

Where PlatformType, AccelType, and CardType parameters are described in the table below. If no parameters are entered, all output source streams are generated to run on a generic x86 platform with no optimizations. Allowable PlatformType, AccelType, and CardType parameters are listed below:

PlatformType	Description	Support Status
none	x86 + Linux	Y
x86	x86 + Linux	Y
ARM	ARM + Linux	N

AccelType	Description	Support Status
none	Plain Intel x86 code, no optimizations	Y
x86	Intel x86 with OpenMP, IPP, other optimizations	Y
ti64	Texas Instruments C64x	N (Partial, but not supported)
ti66	Texas Instruments C66x	Y

CardType	Description	Support Status
none	If not entered, then default CardType containing CPU matching AccelType is used	Y
SIGC6678	Signalogic 32-core C6678 accelerator	Y
SIGC6678-64	Signalogic 64-core C6678 accelerator	Y

Note that it's also possible to run the CIM pre-processor from an arbitrary location, for example:

```
   cimpp -s/install_path/Signalogic/CIM/apps/test_demo/ffmpeg_demo/ -px86 -ati66x -
cSIGC66XX file1.c file2.c
```

8.2 vid_streaming demo

To run the CIM software vid_streaming demo, first cd (change directory) to the vid_streaming folder:

```
   cd /install_path/Signalogic/CIM/apps/test_demo/vid_streaming
```

Then enter:

```
   cimpp -px86 -ati66x -cSIGC66XX vid_streaming.c
```

```
cimpp.sh -px86 -ati66x -cSIGC66XX vid_streaming.c
```

8.2.1 vid_streaming CIM Build Description

Figures 8-1a thru 8-1c below show expected output during the CIM build process for vid_streaming:

```
cimrt$ ./cimpp.sh -s/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/ -px86 -ati66x -cSIGC66XX ffmpeg_demo.c
############################################################
#Running auto build process with following parameters
#
#Main User source file  -  ffmpeg_demo.c
#Number of source files -  1
#Soure file list        -  ffmpeg_demo.c
#Source file path       -  /root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/
#Platform Type          -  x86
#Accelerator Type       -  TI66X
#Card Type              -  SIGC66XX
#CCS project directory  -  /root/Signalogic_2012v4/CIM/cimrt/../../mCPU_DSP/C6xxx/BIOS/
#
############################################################
##################
#Invoking Parser#
##################
Executing ./cimss, first input file is ffmpeg_demo.c!
argc = 5
argv = ./cimss -D_LINUX_ -I/root/Signalogic_2012v4/DirectCore/include/ -I/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo/ ff
mpeg_demo.c
Before calling frontend
new_argv = ./cimss -D_LINUX_ -I/root/Signalogic_2012v4/DirectCore/include/ -I/root/Signalogic_2012v4/CIM/apps/test_demo/ffmpeg_demo
/ ffmpeg_demo.c
Found pragma in SECTION
cim taskc 8, 33333, 1, hostflg, bufrdy
function name: ffmpeg_demo_cim_func_1

/root/Signalogic_2012v4/CIM/cimrt
####################################################
#Creating func_task files and editing _x86.c file#
####################################################
Read [ffmpeg_demo_cim_func_1.c]
 from parser logfile
Read [ffmpeg_demo_x86.c]
 from parser logfile
Currently processing file ffmpeg_demo_x86.c
Generating modified host file with name ffmpeg_demo_x86_rt.c
Region 1 has scalar variables :
  SOCKET hSocket;

About to add array variables in region 1 :
 8 8 8 8 8
##################################
#Starting Iterative build process#
##################################
######################
#Generating Makefile#
######################
######################
#Generating .out file#
######################
rm -rf ffmpeg_demo.out
rm -rf *.obj
rm -rf ./configPkg

Building file: ../cim.cfg
Invoking: XDCtools
/opt/ti/xdctools_3_25_05_94/xs --xdcpath="/opt/ti/bios_6_34_04_22/packages/;/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/packages/;/
opt/ti/ipc_1_24_00_16/packages;/opt/ti/pdk_C6678_1_0_0_17/packages;/opt/ti/ndk_2_22_00_06/packages;/opt/ti/framework_components_3_2
2_00_05/packages;/opt/ti/edma3_lld_02_11_03_02/packages;/opt/ti/xdais_7_20_00_07/packages" xdc.tools.configuro -o configPkg/ -t ti.
targets.elf.C66 -p SigC66xx -b /opt/ti/xdctools_3_25_05_94/packages/xdc/bld/config.bld -c /opt/ti/C6000CGT7.4.2/ ../cim.cfg
making package.mak (because of package.bld) ...
generating interfaces for package configPkg (because package/package.xdc.inc is older than package.xdc) ...
configuring cim.xe66 from package/cfg/cim_pe66.cfg ...
useModule Settings: 6600
Auto-registering resource manager for EDMA3CHAN
Inside EDMA3 RM getLibs
    will link with ti.sdo.edma3.rm:lib/c6678-evm/66/release/edma3_lld_rm.ae66
cle66 package/cfg/cim_pe66.c ...
Finished building: ../cim.cfg

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
```

Figure 8-1a, vid_streaming CIM build process output display

```
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../buildRTAF.lst
[Analog_io.c]
[bsp.c]
[gpio.c]
[host.c]
[logic_uart.c]
[main.c]
[McBSP_Isr.c]
[profile.c]
[main_rtaf.c]
[rman_init.c]
[rtaf_hw_init.c]
[rtaf_periph.c]
[sd4_init.c]
[standalone.c]
[timer1_gpio.c]
[userproc.c]
[Bitrev.c]
[C6xxxLib.c]
[c_memory.c]
[Codec.c]
[event.c]
[Fft2.c]
[fs_conv.c]
[Isr0.c]
[Isr1.c]
[Isr2.c]
[Isr5.c]
[Isr6.c]
[Modul0.c]
[Modul1.c]
[Modul2.c]
[Modul3.c]
[Modul4.c]
[Modul5.c]
[Modul6.c]
[Modul7.c]
[Sintab.asm]
[t6xbeg.asm]
[t6xdef.asm]
[vectors.asm]
Finished building RTAF object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../buildH264.lst
[alg_create.c]
[alg_malloc.c]
[platform_osal.c]
[resourcemgr.c]
[h264hpvenc_ti_buffermanager.c]
[h264hpvenc_ti_config.c]
```

Figure 8-1b, vid_streaming CIM build process output display, cont.

```
[h264hpvenc_ti_mc_api.c]
[h264_encode.c]
[h264hpvenc_ti_testwrapper.c]
[h264hpvenc_ti_GateMPBarrier.c]
[h264hpvenc_ti_edmaConfig.c]
[h264hpvenc_ti_mcIpc.c]
[h264hpvenc_ti_shmemmanager.c]
[siuVigdkGmp.c]
Finished building H264 object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt  -@../cim_build.lst
Finished building cim object files.

Building object files.
Invoking: C6000 Compiler
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 -I/root/Signalogic_2012v4/TI_C6X/Voice/AM
R-WB/ -I/root/Signalogic_2012v4/TI_C6X/Voice/common/ -I/root/Signalogic_2012v4/mCPU_DSP/ -I/root/Signalogic_2012v4/mCPU_DSP/Video -
I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/ -I/root/Signalogic_2012v4/mCPU
_DSP/Voice/ -I/root/Signalogic_2012v4/mCPU_DSP/Host/ -I/root/Signalogic_2012v4/mCPU_DSP/CIM -I/root/Signalogic_2012v4/mCPU_DSP/C6xx
x/ -I/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -I/opt/ti/ndk_2_22_00_06/packages/ti/ndk/inc/ -I/opt/ti/xdctools_3_25_05_94/packa
ges/ -I/opt/ti/bios_6_34_04_22/packages/ -I/opt/ti/bios_6_34_04_22/packages/ti/bios/include/ -I/opt/ti/C6000CGT7.4.2/include/ -I/op
t/ti/C6000CGT7.4.2/lib/ -I/opt/ti/pdk_C6678_1_0_0_17/packages/ -I/opt/ti/xdais_7_20_00_07/packages -I/opt/ti/xdais_7_20_00_07/packa
ges/ti/xdais -I/opt/ti/xdais_7_20_00_07/packages/ti/xdais/dm -I/opt/ti/edma3_lld_02_11_03_02/packages -I/opt/ti/framework_component
s_3_22_00_05/packages -I/opt/ti/framework_components_3_22_00_05/packages/ti/sdo/fc/utils/api -I/opt/ti/ipc_1_24_00_16/packages -I/o
pt/ti/mcsdk_video_2_2_0_38/dsp/components --symdebug:dwarf --display_error_number --diag_warning=225 --mem_model:data=far --abi=eab
i --preproc_with_compile --cmd_file=./configPkg/compiler.opt   ../ffmpeg_demo_func_task_1.c ../ffmpeg_demo_cim_func_1.c
[ffmpeg_demo_func_task_1.c]
[ffmpeg_demo_cim_func_1.c]
Finished building generated object files.

Building target: ffmpeg_demo.out
Invoking: Linker
/opt/ti/C6000CGT7.4.2/bin/cl6x -mv6600 -O3 -D_CIMF -D_BIOS_ -D_SIGC6XXX_ -D_SIGC66XX_ -D_SYSBIOS_ -DCHIP_6678 -DCHIP_C66XX -D_TI66X
 -D_USE_CM_ -DC6600 -D_USE_RMAN_ -D_USE_IPC_ -D_USE_VID_ -D_USE_NDK_ -Dti_targets_elf_C66 --display_error_number --diag_warning=225
 --mem_model:data=far --abi=eabi -z -mffmpeg_demo.map --warn_sections --display_error_number -i/opt/ti/C6000CGT7.4.2/lib/ -i/opt/ti
/C6000CGT7.4.2/include/ -i/root/Signalogic_2012v4/mCPU_DSP/C6xxx/BIOS/ -i/opt/ti/pdk_C6678_1_0_0_17/packages/ -i/opt/ti/pdk_C6678_1
_0_0_17/packages/ti/transport/ndk/nimu/lib/  -i/root/Signalogic_2012v4/TI_C6X/Voice/AMR-WB/ -i/root/Signalogic_2012v4/mCPU_DSP/Vide
o -i/opt/ti/pdk_C6678_1_0_0_17/packages/ti/csl/lib -i/opt/ti/mcsdk_video_2_2_0_38/dsp/components/ti/mas/tsu/ce66 -i/opt/ti/mcsdk_vi
deo_2_2_0_38/dsp/components/ti/mas/util/ce66 --reread_libs --rom_model -offmpeg_demo.out *.obj ffmpeg_demo_func_task_1.obj ffmpeg_d
emo_cim_func_1.obj -l./configPkg/linker.cmd  -llibc.a -lh264hpvenc_ti.le66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/cppi/lib/ti
.drv.cppi.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/qmss/lib/ti.drv.qmss.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/
pa/lib/ti.drv.pa.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/ti/drv/pcie/lib/ti.drv.pcie.ae66 -l/opt/ti/pdk_C6678_1_0_0_17/packages/
ti/platform/evmc6678l/platform_lib/lib/debug/ti.platform.evm6678l.ae66 -lti.csl.ae66 -l../t66linkSYSBIOS.cmd
<Linking>
Finished building target: ffmpeg_demo.out

/root/Signalogic_2012v4/CIM/cimrt
############################
#Generating HOST Executable#
############################
rm -rf *.o
rm -rf ffmpeg_demo
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  -c
 ffmpeg_demo_cim_func_1.c -o ffmpeg_demo_cim_func_1.o
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  -c
 ffmpeg_demo_x86_rt.c -o ffmpeg_demo_x86_rt.o
/usr/bin/gcc -I/root/Signalogic_2012v4/DirectCore/include  -O3 --std=c99 -D_CIM -D_SIGC66XX -D_LINUX_ -D_POSIX_C_SOURCE=200809L  ff
mpeg_demo_cim_func_1.o ffmpeg_demo_x86_rt.o  -lhwmgr -lhwlib -lpthread -lstdc++  -o ./ffmpeg_demo
cimrt$
```

Figure 8-1c, vid_straming CIM build process output display, cont.

After the CIM build process completes, a vid_streaming executable file will be created, and can be run as a typical Linux program:

```
  vid_streaming$ ./vid_streaming -m1 -f1250 -s2 -estream.out -cSIGC66XX-8
-i/install_path/Signalogic/video_files/airshow_352x288p_30fps_420fmt.yuv
-x352 -y288 -r30 -B1500000 -oencoded.h264
```

which should produce screen display output similar to Figure 8-2 below.

```
DSInitCores: qwCoreList = 0xff
SlotIndx = 0
Card_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard:
InitCardT50 success
DSXInitCard Successful!
Loading input video .yuv data...
Clearing output video mem region...
Loading cores, core list = 0xff
DDR3 Time lapsed in microseconds : count 11878204  12
L2SRAM Time lapsed in microseconds : count 920  8
MSMCSRAM Time lapsed  in microseconds : count 740   11
DSXRunCard Successful!
Core 1 Finished
Core 2 Finished
Core 3 Finished
Core 4 Finished
Core 5 Finished
Core 6 Finished
Core 7 Finished
Core 0 Finished
Reading output video data, saving to file...
C66x testrun[0, 0x85f95c] = 9003
C66x cores error code[0x85ec28] = 0
numFramesEncoded[0x8597b4] = 40
vid_streaming$
```

Figure 8-2, vid_streaming demo output display

Note that on the vid_streaming folder, only the files vid_streaming.c and zero.dat are required. Except for the "encoded.h264" output file, which may be created when the demo executable code is run (as shown in Figure 8-2 above), all other files on the vid_streaming folder are auto-generated by the CIM build process.

8.3 Convolution demo

The CIM convolution demo is compatible with x86 OpenMP, c66x OpenMP, and the CIM® Hyperpiler™, which also uses OpenMP pragma syntax. The convolution demo is intended to (i) give a simple example of using CIM, (ii) show differences between the three platforms, and (iii) explain the theory behind the CIM Hyperpiler.

The CIM Hyperpiler generates source code streams for both x86 and c66x in order to build – from a single high level C/C++ program -- heterogeneous core run-time executables that coordinate and communicate with each other. Below is a source code excerpt from conv.c; the area marked in red shows use of #ifdefs to allow cross-platform compatibility.

```
int x[X_LEN + 2*H_LEN - 2] = { 0 };
int h[H_LEN] = { 0 };
int y[X_LEN + H_LEN - 1] = { 0 };

int h_len = H_LEN;
int x_len = X_LEN + 2*H_LEN - 2;
int conv_len = X_LEN + H_LEN - 1;

int sum;
int i,j;

int main (int argc, char *argv[]) {
/* use platform ifdef's to determine which OpenMP pragma keyword to use */

   #ifdef _CIM
   #pragma cim parallel for num_threads 5 private(i,j,sum) /* CIM combined x86 and c66x
   #else
   #pragma omp parallel for private(i,j,sum) /* x86 only or c66x only */
   #endif

   for (i=0; i<conv_len; i++) {  /* time domain convolution as 2-level for-loop */

      for (j=0; j<h_len; j++) sum += h[j] * x[i + j];

      y[i] = sum;
   }
/* Measure and report profiling results */
```

Figure 8-3, Convolution demo source code, with cross-platform compatibility #ifdefs highlighted

To run the CIM software convolution demo, first cd (change directory) to the conv folder:

```
cd /install_path/Signalogic/CIM/apps/test_demo/conv
```

Then enter:

```
make cim
```

Note that giving make with a cim target runs the following three (3) steps:

- CIM Hyperpiler process
- target CPU make
- standard Linux x86 make

Here is what the cim target looks like inside the Makefile:

```
./../../../cimrt/cimpp -px86 -ati66 -cSIGC66XX-32 -s
/root/Signalogic/CIM/apps/test_demo/conv/conv.c
```

Next, to run the convolution demo, enter:

```
./conv -m0x1f -f1400 -cSIGC66XX-8
```

The core list (`-m`), clock rate (`-f`), and card designator (`-c`) command line options are the same as described in DirectCore demo program sections above (e.g. boardTest, fftTest, streamTest, etc).

 Copyright © Signalogic, Inc. 2012 - 2016

Here are some notes about running CIM generated executables:

1. Target CPU executables are automatically downloaded, initialized, and run. There is no ".out" file or other executable file entry in the command line (unlike DirectCore test programs that are run manually)

2. If you give a standard

   ```
   make
   ```

   command, then you are re-running the Linux make, not re-running the CIM process and regenerating x86 and target CPU source codes.

3. When giving the card designator in the command line, if you are running multiple process or sharing the card with other users, then it's necessary to pay attention to the number of cores requested. For example, the convolution demo command line above has the following card designator entry:

   ```
   -cSIGC66XX-32
   ```

   which requests 32 cores. If you enter:

   ```
   -cSIGC66XX-8
   ```

   then only 8 cores are requested

Figure 8-4 below is a screen grab showing expected output from the convolution demo.

 Copyright © Signalogic, Inc. 2012 - 2016

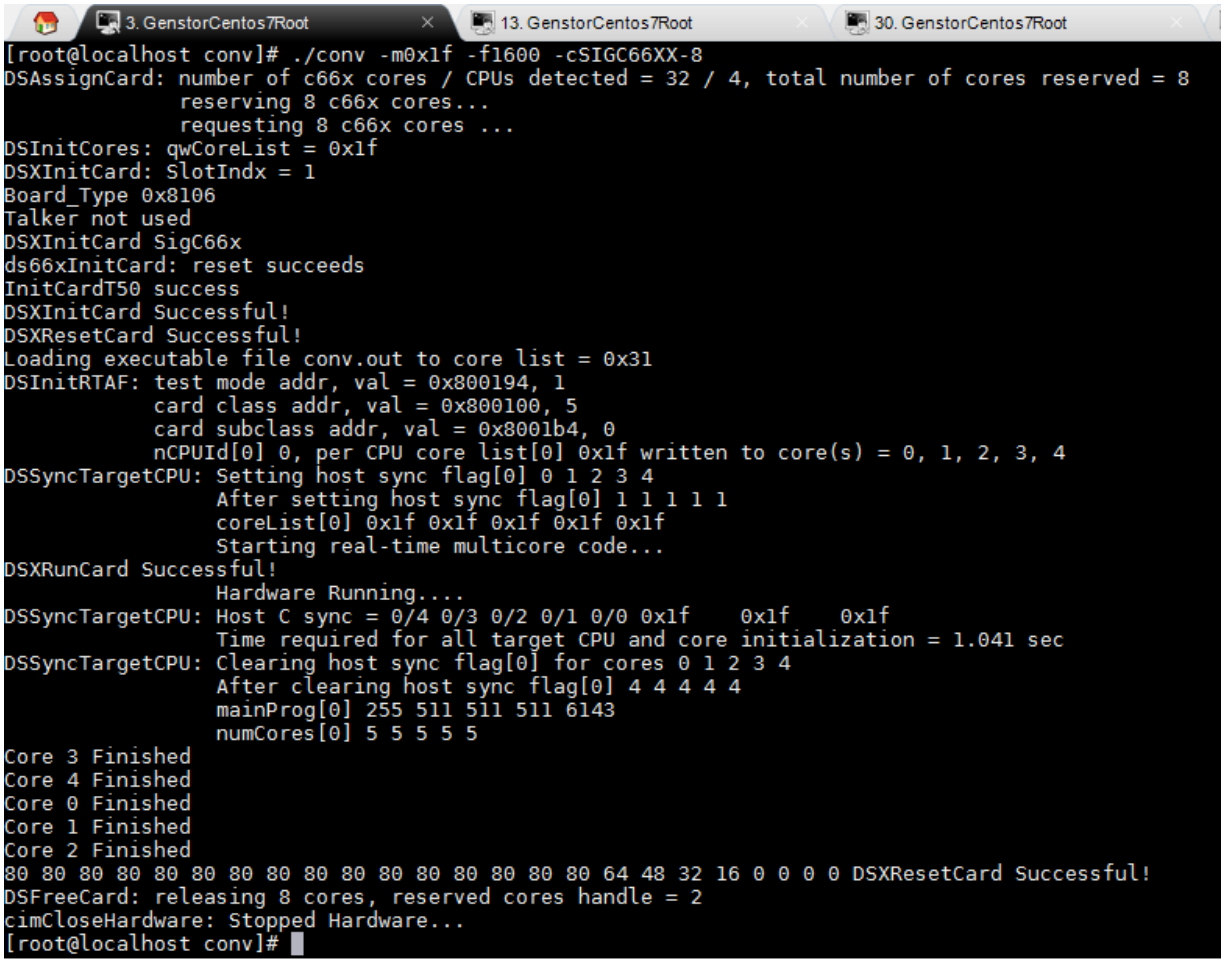
3. GenstorCentos7Root
13. GenstorCentos7Root
30. GenstorCentos7Root
[root@localhost conv]# ./conv -m0x1f -f1600 -cSIGC66XX-8
DSAssignCard: number of c66x cores / CPUs detected = 32 / 4, total number of cores reserved = 8
reserving 8 c66x cores...
requesting 8 c66x cores ...
DSInitCores: qwCoreList = 0x1f
DSXInitCard: SlotIndx = 1
Board_Type 0x8106
Talker not used
DSXInitCard SigC66x
ds66xInitCard: reset succeeds
InitCardT50 success
DSXInitCard Successful!
DSXResetCard Successful!
Loading executable file conv.out to core list = 0x31
DSInitRTAF: test mode addr, val = 0x800194, 1
card class addr, val = 0x800100, 5
card subclass addr, val = 0x8001b4, 0
nCPUId[0] 0, per CPU core list[0] 0x1f written to core(s) = 0, 1, 2, 3, 4
DSSyncTargetCPU: Setting host sync flag[0] 0 1 2 3 4
After setting host sync flag[0] 1 1 1 1 1
coreList[0] 0x1f 0x1f 0x1f 0x1f 0x1f
Starting real-time multicore code...
DSXRunCard Successful!
Hardware Running....
DSSyncTargetCPU: Host C sync = 0/4 0/3 0/2 0/1 0/0 0x1f 0x1f 0x1f
Time required for all target CPU and core initialization = 1.041 sec
DSSyncTargetCPU: Clearing host sync flag[0] for cores 0 1 2 3 4
After clearing host sync flag[0] 4 4 4 4 4
mainProg[0] 255 511 511 511 6143
numCores[0] 5 5 5 5 5
Core 3 Finished
Core 4 Finished
Core 0 Finished
Core 1 Finished
Core 2 Finished
80 80 80 80 80 80 80 80 80 80 80 80 80 80 80 64 48 32 16 0 0 0 0 DSXResetCard Successful!
DSFreeCard: releasing 8 cores, reserved cores handle = 2
cimCloseHardware: Stopped Hardware...
[root@localhost conv]#

Figure 8-4, Convolution demo output screen capture